A Learning Approach for Extending Human-Robot Collaboration to Manufacturing-Specific Tasks

Parker Owan

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Santosh Devasia, Chair

Joseph Garbini, Chair

Ashis Banerjee

James Buttrick

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

A Learning Approach for Extending Human-Robot Collaboration to Manufacturing-Specific Tasks

Parker Owan

Co-Chairs of the Supervisory Committee:
Professor Santosh Devasia
Department of Mechanical Engineering

Professor Joseph Garbini
Department of Mechanical Engineering

This thesis presents the development and evaluation of methods for extending shared autonomy to limited-access manufacturing telerobotics. Shared teleoperation has potential to reduce strenuous working conditions and increase process efficiency in this application domain. However, current methods for shared autonomy in such applications are limited by: (**Q1**) difficulty handling pose errors that arise from uncertain placement of the manipulator; (**Q2**) fragility to off-nominal situations that have potential to degrade system performance; and (**Q3**) difficulty automating the physical tasks prevalent in limited-access operations.

The main contribution of this thesis is an imitation learning method that produces dynamical models of a manufacturing task in order to address these limitations. The method (i) learns a structured model of the data, including positions, velocities, accelerations, and forces; (ii) performs a state-action decomposition of the model; and (iii) constructs dynamical models to describe the motion and forces for each action, as well as the sequence of actions. The resulting model of the task dynamics enables the following contributions. (**C1**) To address **Q1**, this work uses motion and force feedback data during human teleoperation to localize the target position for the task, and trades control between human and autonomy based on localization uncertainty. The challenges are how to produce a reliable estimate

within the required tolerance to enable automation, and how to handle human-robot disagreements that arise due to estimate uncertainty. Use of the task dynamics model addresses the first challenge by providing a sufficient likelihood for observed positions and accelerations during task teleoperation, and a procedure is developed to address the second challenge. (**C2**) To address **Q2**, this work trades control to the human in off-nominal situations. The challenge is how to detect off-nominal situations. Use of the task dynamics model addresses this challenge by providing an expectation of interaction forces during task automation. (**C3**) To address **Q3**, this work uses imitation learning to develop a control policy for automation that mimics an expert operator. The challenge is how to imitate demonstrations that are not classical reaching movements, i.e. there is no clear target state. The developed task dynamics learning approach automates the isolation of target states to develop the sequence of actions in between them, thus providing reference motions and forces to which the system can be regulated during task automation.

The final contribution (**C4**) experimentally evaluates the methods for an aerospace-manufacturing hole cleaning task. (**E1**) Use of control trading based on localization estimate uncertainty from **C1** reduces completion times for a hole locating task by 50% as compared with teleoperation. (**E2**) Use of kinetics predictions to address off-nominal situations in assisted teleoperation from **C2** reduces completion time by 17% and operator forces by 68% as compared with assistance without the method. (**E3**) Use of kinetics predictions from **C3** as feedforward commands reduces tracking errors in position (61%), velocity (57%), and force (53%), as compared with feedback compensation alone during task automation. (**E4**) In concert, the contributions enable shared autonomy in a user study (n=8) to reduce completion time by 54.0%, operator energy expenditure by 80.5%, and operator forces by 44.0% as compared with teleoperation. These results illustrate the potential of the thesis contribution to improve process efficiency and mitigate strenuous work conditions for a class of manufacturing tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

SHARED AUTONOMY: a paradigm where human and robot actions are combined in order to collaboratively achieve a common goal.

LIMITED-ACCESS: a class of manufacturing conditions characterized by tight work spaces and difficult access.

CONTROL POLICY: a mathematical mapping from state to action, defined over the domain of possible states.

COMPLIANT POLICY: a control policy that encodes compliance within the state to action map, i.e. the stiffness with which the policy operates about an equilibrium state.

KINEMATICS: the study of motion, i.e. accelerations, velocities, and positions

KINETICS: the study of the forces that cause motion

TASK DYNAMICS: a model of the kinematics and kinetics for a robot tool to complete a task, e.g. bottle brushing operation

IMITATION LEARNING: an approach to develop a policy from observations of human behavior, i.e. by encoding the state to action map of the human demonstration into a control policy.

AUTONOMY: a control policy that regulates a set of task dynamics produced via imitation learning.

HAPTIC SHARED CONTROL: is the application of autonomy via a force applied to the haptic teleoperation device.

OFF-NOMINAL: situations that diverge from conditions under which a control policy has been developed.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Professor Santosh Devasia for bringing me to the University of Washington, and cultivating an environment where I could see myself grow daily. His challenging questions, discussions on what it means to be a Doctor of Philosophy, and our many in-depth explorations into details of the research were core to my development throughout my studies and have changed how I solve problems.

I would also like to express my sincere gratitude to my co-advisor Professor Joseph Garbini for his insightful and challenging discussions that would help me to refactor my approach to problems both in depth and related to the bigger picture. His constant search for clarity and how research relates to other studies of science serves as a lasting model for my pursuits.

I would like to thank the rest of my committee, Professor Sidd Srinivasa, Professor Ashis Banerjee, and James Buttrick for their invaluable insight, opportunities, and expectations. Their support has a lasting influence on my career.

I wish to express my appreciation to Jim Buttrick, Professor Devasia, and Chair Reinhall for extending to me a unique opportunity to work alongside both industry leaders and premier researchers who are advancing the edge of science with solutions to real problems.

I wish to thank the team at my lab for their challenging discussions and encouragement, sometimes over a drink at the faculty club or the occasional weekday ski trip.

I want to thank my wife, Lyndsey, for her dedication, patience and willingness to support me throughout such a demanding experience.

Lastly, I want to thank my father Chris, for being a model to which I can aspire; my brother Cory, for sharing my ambitions and supporting my accomplishments throughout my studies; and my mother Andrea, for getting me started on this track from day one.

DEDICATION

For Chris and Lyndsey.

Chapter 1
INTRODUCTION

1.1 Motivation

This work aims to increase process efficiency and reduce mechanic workload through the use of robotic shared teleoperation. Limited-access work is characterized by physical work space restrictions, physical operations (e.g. fastening, drilling, cleaning), process inconsistencies, and partially-known environments.

1.1.1 Barriers to limited-access manufacturing

Consider the example limited-access setting shown in Fig. 1.1, involving a sequence of mating, drilling, and cleaning operations. In the final task of the sequence, a bottle brush is used to remove debris leftover from the hole drilling process. This final cleaning task in the manufacturing sequence serves as a case study for the experiments performed in this research. To date, such limited-access work is still primarily a manually-performed operation in aerospace manufacturing for the following reasons:

- **Work space geometries make physical access difficult.** Work space restrictions, such as tight access points, narrow corridors, and obstacles, make it physically challenging for a mechanic to access the region where work is performed. This (i) increases potential for ergonomic and health issues, especially if the time spent in the space is significant, and (ii) adversely impacts mechanic performance by increasing task completion times. The unique geometries associated with limited-access spaces prohibit the use of many commercially available robots, driving the need for custom kinematic solutions that are manually placed into the workspace [**?**].

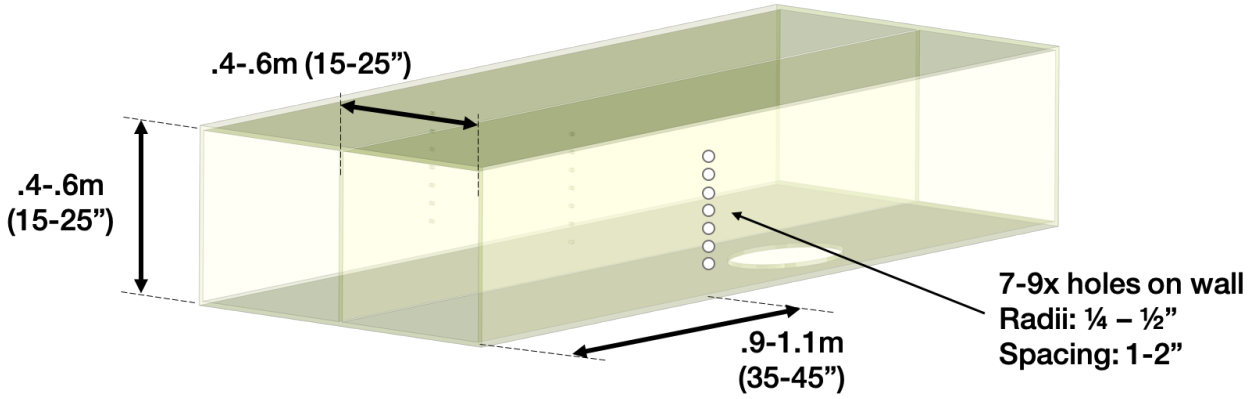


Figure 1.1: An example limited-access manufacturing application involves work on a set of holes roughly 1 meter inside a narrow space. The application is defined by a sequence of tasks: i) a mating part is inserted into the space; ii) pilot holes are drilled out and several more holes are added using a template; iii) fine and course surface finishing are performed to clean up the drilled holes; and iv) a bottle brush is used to remove remaining debris.

- **Process variations increase the complexity of automation.** Limited-access manufacturing, especially for aerospace, exhibits variable conditions from part to part. At any given stage in the assembly, the work piece has potential to differ from (i) the work piece at other stages in the assembly, (ii) a previous work piece at the same stage in the assembly, and (iii) the designed computer representation. The result is that mechanics must be contextually aware of the current condition of assembly and be able to adjust their process based on such part variations. Full automation is infeasible due to the work space limitations on placing external sensor arrays that would be needed to handle these uncertainties.

In summary, the complexities required for automation to handle the uncertain processes and work space restrictions found in limited-access manufacturing render fully autonomous solutions impractical.

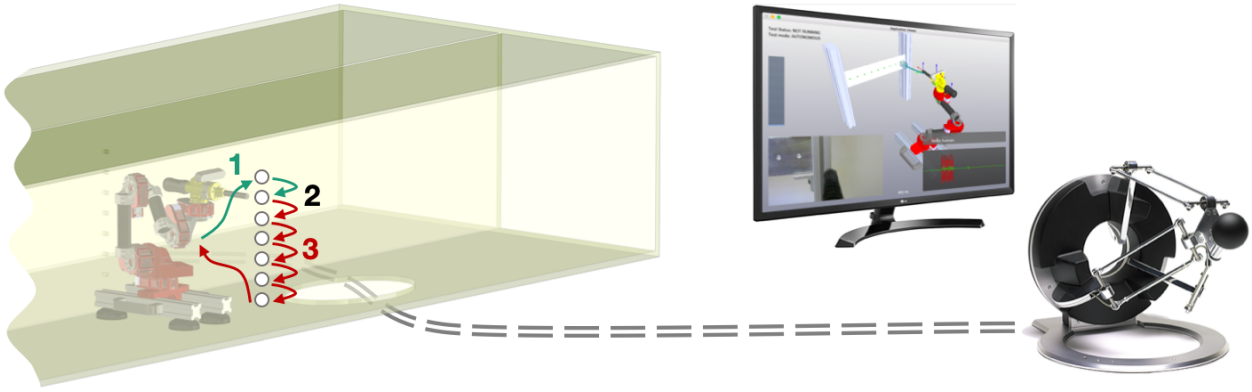


Figure 1.2: The proposed system includes a (a) lightweight robot that is hand-placed in the limited-access workspace, and (b) a workstation to teleoperate the robot via a haptic device and graphical interface. The proposed collaborative procedure involves (1) the human teleoperating the cleaning procedure for the first few holes, (2) the robot gaining confidence in states inferred from human actions, and (3) the autonomy completing the operation.

1.1.2 Shared autonomy in robotic teleoperation

This work considers the use of shared autonomy in teleorobotics to address limited-access process variability. Such a human-robot collaboration balances automation speed and repeatability with human reasoning and adaptability to improve the overall system efficiency and robustness to uncertainty. The prototype system shown in Figure 1.2 is a lightweight robot arm that is manually-placed close to the task, enabling the human to perform the task via teleoperation without having to repeatedly crawl into the tight workspace. In nominal operation, the shared autonomy is comprised of three steps: (1) the human begins the task (e.g. brush cleaning) while the autonomy infers target features (e.g. hole target location) needed for automation from human actions and onboard sensor feedback; (2) once the autonomy gains sufficient confidence in target estimates, it (3) takes control from the human and the autonomous policy completes the operation.

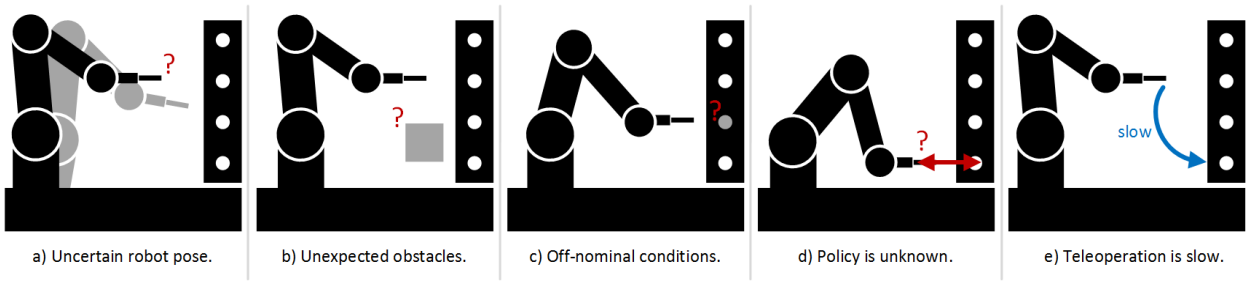


Figure 1.3: The proposed system introduces research questions: (a) how to reduce pose uncertainty that stems from manual placement, (b-c) how to detect and resolve off-nominal situations that arise from environment uncertainty, and (d) how to automate the manufacturing tasks. If these questions can be addressed, then shared autonomy can be employed to improve performance over teleoperation, which is known to be slow (e).

1.2 Problem Statement and Research Questions

Problem statement: Can human actions be used to increase the level of autonomy?

The following research questions (Figure 1.3) arise from the problem statement:

Q1 Localization: *How to reduce pose uncertainty stemming from manual placement?*
In order for autonomy to be possible, the robot must be able to localize to the work (Figure 1.3 a) without the use of lengthy calibration procedures or sensor arrays that would prohibitively impact process efficiency and result in disuse of the technology. Fortunately, human teleoperation actions contain information that can indicate where the robot is relative to the work. In this manner, the robot pose can be recovered from human intention estimation [**?**, **?**, **?**]. However, solving this problem is non-trivial for the following reasons:

- **Information ambiguity limits estimate error convergence.** State of the art estimation techniques [**?**, **?**] rely on uncertainty estimates to drive the rate of convergence, trading-off belief in estimate versus belief in evidence (observed

data). If the observation uncertainty is too high, the estimate will not converge to the required tolerance for sufficiently confident insertion of the brush into the hole. Therefore, accurate *probabilistic* predictions of human actions are needed in order for teleoperator motion to provide localization capability.

- **Estimate errors can introduce human-robot disagreement.** If the robot is overconfident or under-confident in target localization due to model limitations and data noise, there is potential for conflicts in control initiative to arise, e.g.: the human attempts to take control because the robot is acting on an overconfident estimate. A method is needed to systematically handle such potential conflicts.

Q2 Off-nominal: *How to detect and resolve off-nominal situations?*

Since teleoperation is known to be slow in task-specific situations [**?**] (Figure 1.3 e), haptic assistance (an intermediate level of autonomy) can aid the teleoperator, resulting in reduced time and operator effort to complete a task. Current methods for teleoperation assistance use the variance of demonstration [**?**, **?**] to proportionally scale haptic compliance around a nominal trajectory. This has potential to negatively impact performance when the operator must oppose the assistance in off-nominal situations (Figure 1.3 b-c), especially in regions of low variance, and thus high stiffness. Addressing this problem is non-trivial for the following reasons:

- **Detection.** Interactions between human and automation occur at the force level in haptic shared control [**?**]. In this paradigm, conflicts arise when the human and autonomy exert opposing forces onto the haptic device [**?**]. Because the human and autonomy are communicating via application of force in haptic shared control, these conflicts are apparent at the force level. The challenges lie in determining a nominal force for comparison and separating noise from truly off-nominal situations.

- **Resolution.** Itoh et al [**?**] discuss the differences between conflicts stemming

from human error versus automation errors. The appropriate course of action is contingent on which agent is the expert in the situation, as well as the application being examined.

Q3 Policies: *How to automate manufacturing tasks?*

To further reduce completion time and operator effort, a control policy is needed to facilitate fully automated task completion (Figure 1.3 d). An ideal policy is intuitively developed, and supports both shared control and full automation. Imitation learning *dynamical systems* [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**] accomplishes these objectives. Most dynamical system approaches focus on reaching movements, which do not consider task kinetics since these are expected to be zero during reaching. However, kinetics are essential in the the class of tasks in limited-access manufacturing, as interaction forces dictate the work that is being performed by the motion. Extending dynamical system approaches to manufacturing tasks is non-trivial for the following reasons:

- **Multimodal dynamics.** Manufacturing tasks (e.g. cleaning, fastening, surface removal) exhibit switching dynamics, such as friction hysteresis in brush cleaning. In general, these tasks may not have a simple one-to-one map from system state to expected forces. Averaging methods [**?**, **?**] such as Gaussian process regression (GPR) [**?**] or Gaussian mixture regression (GMR) [**?**] may learn poor predictions of these kinetics if the modality is not explicitly addressed.

- **Unknown target states.** Dynamical approaches often employ a single target state, which for stability purposes can be equilibrium points of the system. However, manufacturing tasks such as brush cleaning exhibit data that has no equilibrium states in the 2nd order representation (i.e. there is no point which is simultaneously zero velocity and zero acceleration) during the task.

1.3 Contributions

The main contribution of this thesis is the development of an imitation learning method to produce a task dynamics model (autonomous dynamical motion and interaction force predictions) from expert demonstrations. The method begins by clustering the demonstration data using Gaussian mixture models (GMM). Next, a subset of clusters are identified as target states by examining motion divergence from the cluster mean in the local data. A set of primitive actions (i.e. sets of clusters) are found by examining the cluster traversal between target states. Then, for each action, second order autonomous dynamical motion and force observation predictions are produced using Gaussian mixture regression (GMR), as in [**?**, **?**, **?**]. Finally, an autonomous system learns to predict the sequence of actions, using a nonlinear autoregressive exogenous (NARX) neural network. This model of task dynamics addresses the research questions (Q1-3) with the following contributions:

C1 *Target localization using task dynamics models.*

To address the challenges in **Q1**, this contribution employs the probabilistic model of task dynamics to provide motion predictions that achieve the needed error convergence to confidently automate the remaining tasks. In the event that human and automation are in conflict over which agent should have control, e.g.: if the localization is overconfident in its estimate, this contribution presents a method for systematically trading control and enacting contingency procedures (*consensus-based traded control*).

C2 *Addressing off-nominal situations.*

To address the challenges in **Q2**, this contribution develops a method to detect off-nominal situations during teleoperation assistance through use of a compliant task dynamics model to predict human-robot interaction forces. The same method is applied to detecting off-nominal situations during full automation through use of the model to predict robot-workpiece interaction forces.

C3 *Autonomous policies for physical tasks.*

To address the challenges in **Q3**, this contribution develops the task-specific dynamics model by decomposing the task into states and actions, providing a set of autonomous dynamical models that are used to predict both kinetics and second-order motion. Optimized trajectories over these models provide reference motions and feedforward interaction forces that are used to regulate robot motion during task automation.

C4 *Experimental evaluations.*

This contribution experimentally evaluates the previous contributions for the remote hole-cleaning task.

- **E1** *Localization.* This experiment presents performance improvements when consensus-based traded control from **C1** is used to resolve disagreements in initiative that arise due to errors in the localization estimate. Results show that the method enables traded control to (i) reduce completion time by 50% as compared with teleoperation, and (ii) increase task success rate from 3% in pure automation to 97% when localization failure modes are introduced.

- **E2** *Off-nominal.* This user study (n=11) presents performance improvements when **C2** is used to address off-nominal situations introduced during shared teleoperation. The proposed method reduces completion times by 17% and teleoperator forces by 68% as compared with shared teleoperation that does not address off-nominal situations.

- **E3** *Policies.* This experiment shows that use of a force feedforward from the model prediction reduces tracking errors in position by 61%, velocity by 57%, and force by 53%, as compared with motion feedback correction alone when the task is fully automated through use of the control policies in **C3**, illustrating the need for kinetics models to improve task automation.

E4 *Demonstration.* This experiment demonstrates the main contribution applied to cleaning the series of nine holes in a user study (n=8). Results show that the contributions **C1-3** enable shared autonomy to reduce completion time by 54.0%, operator energy expenditure by 80.5%, and operator mechanical forces by 44.0%, as compared with pure teleoperation without the proposed shared autonomy methods.

1.4 Outline

The following summarizes the topics discussed in each chapter of the thesis.

Chapter 2: ***Background.*** This chapter reviews the state of the art in techniques for enabling shared autonomy. Focus is given to work in imitation learning for control policies, addressing off-nominal situations, localization, and methods for shared and traded control. This chapter also discusses tools that are used in the contributions.

Chapter 3: ***Control Trading Based on Localization Confidence.*** This chapter discusses mixed-initiative traded control from confidence in localization estimates. The chapter presents a method (**C1**) for resolving disagreement between agents when localization failure modes arise. Particle filtering techniques for localization and confidence bounds are developed based on task geometry. An experiment (**E1**) examining conflicts from computer vision errors is used to test the proposed method. The work in this chapter was presented at the IEEE Conference on Advanced Intelligent Mechatronics (AIM 2017) [**?**].

Chapter 4: ***Managing Off-Nominal Situations.*** This chapter presents a method (**C2**) that uses interaction forces, in this case between the robot and the human, to determine if the system has entered an off-nominal condition. A user-study experiment (**E2**) examines performance improvements in shared teleoperation through use

of the proposed method. The work in this chapter was presented at the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS 2018) [**?**].

Chapter 5: ***Learning Policies for Task Automation.*** This chapter details the development of policies for task automation (**C3**). The chapter derives the task dynamics model structure, and proposes a procedure to learn this model. A method for model selection is discussed. A trajectory optimization technique for finding policies over the learned model is presented.

Chapter 6: ***Experiments for a Hole Cleaning Task.*** This chapter presents experiments to complete **C4**. Results of a tracking experiment (**E3**) show the reduction of tracking errors through use of the kinetics estimates from **C3**. The chapter concludes with a set of human subject experiments to demonstrate the benefits of the proposed shared autonomy for an exemplar manufacturing task (**E4**).

Chapter 7: ***Conclusions.*** This chapter provides a summary of the technical contributions, limitations, and a discussion of future work.

Chapter 2
BACKGROUND

2.1 Shared autonomy

This thesis develops contributions within the study of shared autonomy, where human and autonomous robot actions are combined in order to collaboratively achieve a common goal. The *level of autonomy* [?, ?, ?, ?, ?] describes the degree to which a system combines human and robot inputs. The following levels of autonomy are discussed or employed in this thesis.

Teleoperation. At one extreme is complete manual control, e.g. through a joystick or haptic interface. In this thesis, a force-reflecting device is used to actively impart forces on the operator during teleoperation of the remotely-located robot. Specifically, this thesis employs bilateral teleoperation [?, ?, ?, ?, ?] to couple both the *flow* (i.e. velocities) and *effort* (i.e. forces) between the haptic input device and robot through a software impedance law. In this way, the user "feels" interaction forces from robot dynamics and contact with the environment through the stiffness gains coupling the haptic device to the robot motion. Primary considerations for the design of bilateral teleoperation systems include interaction realism through transparency [?, ?], and stability in the presence of communication or sampling rate time delays [?, ?, ?, ?]. Teleoperation is used as a baseline comparison in each of the experiments examined in this thesis.

Safe mode. In safe mode, the user teleoperates the robot while the autonomy reactively exercises precautions in the presence of detected potential hazards [?, ?]. This is useful to protect the robot and alleviate the mental burden on the user of having to repeatedly handle such situations. This level is found in assistive wheelchairs [?, ?], remotely operated mobile

robots for search and resuce (SAR) operations [?], and manipulators [?]. Although the work in this thesis does not explicitly implement safe modes, a method to employ machine initiatives that override the user in the presence of known problems is a contingency discussed in Chapter 3.

Shared control. In shared control, the user teleoperates the robot while the autonomy actively provides assistance to aid the user in a task [?, ?, ?, ?, ?, ?, ?, ?, ?, ?]. This mode is useful when the autonomy is not capable of performing the task independent of the human, or when both the autonomy and robot are needed for a task, e.g. moving a table [?]. Haptic shared control [?, ?, ?, ?, ?, ?] is a specific case of shared control where assistance is provided through the addition of virtual reflected forces in the haptic teleoperator interface. This thesis employs haptic shared control for assistance in Chapter 4, developing force-control policies from user demonstrations as in [?, ?].

Full automation. At the other extreme of shared autonomy is full automation, where the robot operates independent of any human input, which is useful for repeated operations where consistency and speed are needed, e.g. manufacturing [?, ?, ?, ?]. In some applications, e.g. low volume small manufacturing entities [?], the engineering investment needed to develop full automation can be too costly or lengthy. To address this challenge, this thesis employs imitation learning [?, ?], where automation learns a *control policy* that imitates a task that has been demonstrated by the human on the robot. This thesis presents a method for learning manufacturing-specific control policies from demonstration to enable full automation in Chapter 5.

Dynamic shared autonomy

The level of autonomy can be dynamically assigned for different portions of a task [?, ?, ?], to balance automation speed with human adaptability when needed. Such dynamic autonomy is desirable because the uncertainty in automated actions evolves over the course

of a task, e.g. due to dynamic operator intentions [?, ?] or variable uncertainty in the developed autonomy [?, ?].

Traded control. This thesis employs traded control [?, ?, ?, ?, ?, ?], where the level of autonomy is dynamically switched between teleoperation and full automation. In mixed-initiative interactions [?, ?, ?, ?, ?, ?, ?], the human and machine jointly manage when this switch occurs. A central challenge in mixed-initiative approaches is how to resolve conflict between the human and machine, e.g.: when both the human and machine seek control of the system. To address this issue, this thesis develops *consensus-based traded control* in Chapter 3, where predefinied contingency procedures are enacted during cases of conflict.

Confidence-based initiative. Confidence in autonomous actions is often used to determine when the robot requests control. Confidence in predicted goals or trajectories has been used to arbitrate blended shared control [?, ?] or influence the stiffness of haptic guidance in shared teleoperation [?, ?]. This thesis similarly uses automation confidence (i) to determine the automation initiative in traded control in Chapter 3, and (ii) to determine the degree of haptic assistance in shared control in Chapter 4.

2.2 Localization from human intention estimation

Localization and mapping

To facilitate rapid manual placement of the robot into the workspace, the robot must localize to the workspace without the use of lengthy calibration procedures that would prohibitively impact process efficiency. Pose localization using visual fiducials [?] is limited by (i) the need for known fiducial locations in the work space and (ii) the need for object detection and recognition to detect a generic fiducial. A challenge with such visual localization methods for limited-access manufacturing are extreme lighting conditions and reflective surfaces such as aluminum that make reliable object recognition difficult without large training datasets. Unfortunately, such large in-situ training datasets are not immediately available

since limited-access work remains manually performed. Localization from point clouds (e.g. from a Kinect-style depth camera) suffer from similar data collection problems.

To relax the need for known workspace feature locations, simultaneous localization and mapping (SLAM) [?,?,?,?,?,?,?,?,?] has been developed to recurrently map feature locations and localize the robot to the mapped features. Sparse SLAM still requires object detection for localization to relevant objects as opposed to just visual features (e.g. SURF, SIFT), requiring large training datasets, and thus suffering from the same issues as fiducial-based localization for the manufacturing application.

Dense SLAM relaxes the need for object detection by mapping and localizing to an entire point cloud, however this introduces a an additional computing requirement to fuse dense sequences of maps. Some dense methods are known to exhibit degraded performance from repetitive features such as long corridors or flat surfaces[1]. These types of surfaces are prevalent in limited-access workspaces, e.g. arrays of evenly-spaced pilot holes or flat geometries. Further, the types of active sensor arrays needed to resolve depth maps in the low light limited-access environments exhibit degraded performance due to infrared scattering from the manufacturing materials present in the workspace. Rather than focus on solving the limited-access perception issue, this thesis considers an alternative approach using human actions to address uncertainties arising from this information deprivation.

Human intention estimation

To address the computational and sensing burden with traditional localization approaches, this thesis proposes the use of sensor feedback during human motion to infer robot pose information. Because the robot is fixed to the workpiece, the pose is assumed static over the duration of the task. The relationship between robot pose and tool position is known from kinematics and real-time joint position feedback. Therefore, by estimating the target

[1]These methods typically use the iterative closest points (ICP) algorithm–an expectation maximization-style optimization that matches points between successive scans–which performs best when there are many unique features to match.

state (e.g. pilot hole location for the bottle-brush operation) during human motion, the robot pose relative to the target can be readily inferred. This thesis utlizes human intention estimation [?, ?, ?, ?, ?, ?, ?] to accomplish target inference from human motion. This thesis builds on three methods for actively inferring human intention.

Hauser [?] proposed a freeform task inference engine (FTIE) using Bayesian inference with Gaussian mixture regression to predict 2D cursor movements on a graphical screen display to improve user experience in graphical teleoperation. The thesis contribution extends the use of Gaussian mixtures and Bayesian inference to the case of recovering the static target state pose during task execution from developed models of human motion.

Wang et al [?] propose the intention driven dynamics model (IDDM) to predict strategies and trajectories in human-robot table tennis. The intention is inferred from dynamics and observation models using Gaussian process regression (GPR). The Bayes filter is solved approximately due to the computational complexity of GPs. This thesis uses this intention-driven dynamics concept but alternately uses Gaussian mixture regression (GMR) to model motions, facilitating nonlinear sampling approaches such as particle filters due to reduced computational cost. This allows a sequence of target poses to be estimated from a multimodal distribution, allowing the localization of the robot pose.

Bayesian filtering

This thesis makes extensive use of the Bayes filter [?, ?]. Recursive Bayesian filtering probabilistically resolves a dynamics model (the "transition") with a sensor model (the "likelihood"). The filter operates on the Markov assumption that the next state in the dynamics model is a function of the current state (in discrete time). Using this assumption, each recursion of the filter involves dynamics prediction and feedback correction steps to resolve the two models. The resulting estimate is the "posterior belief" in the state. The filter must be initialized with a "prior belief" in the state, often simply chosen as a uniform distribution over the domain.

The filter applies to discrete, continuous, and mixed discrete-continuous state variables

found in Dynamics Bayesian networks (DBNs). If the state is continuous, and the models are linear and zero-mean Gaussian, the Kalman filter is an exact solution to the Bayesian estimator. For nonlinear/non-Gaussian models, the extended Kalman filter (EKF) is commonly used, however this performs poorly when the true posterior is multimodal since it performs a linearization and Gaussian fit. Ensemble Gaussian filters [**?**] have been used to address the modality problem.

In this thesis, sufficient localization accuracy is only achieved with nonlinear likelihood models. To relax the linear and Gaussian assumptions of the previously mentioned filters, a sequential monte-carlo sampling (SMC) technique, i.e. the particle filter (PF) [**?**, **?**, **?**, **?**, **?**] is employed. Because monte carlo methods are computationally expensive (N simulations each time step for N particles), simple models and a low number of particles N are desirable, however more particles tend to better approximate the posterior density[2]. A known problem with particle methods is sample degeneracy, where after some iterations only a small number of particles carry the estimate. This can be addressed by the addition of resampling when the degenerate condition is detected, which is utilized in the particle filters in this thesis.

2.3 Addressing off-nominal situations in teleoperation

Off-nominal situations have potential to cause conflicts in human-robot collaboration because they degrade the prediction capability of the autonomy. It has been proposed [**?**] that the ability of the robot and human should influence which agent is granted final authority. One way to implement this is by limiting the maximal amount of haptic assistance [**?**], however this increases the operator workload. In this thesis, it is assumed that the operator is skilled in the task, and that conflicts arise when the collaborative system experiences off-nominal situations that the assistance has not yet been developed to handle. In this manner, human actions can be used to inform the detection of off-nominal situations.

In shared teleoperation enabled by imitation learning [**?**, **?**], variance of the demonstration

[2]The Kullback-Leibler divergence [**?**] has been used to adapt the number of particles N by evaluating the information content of each particle.

influences haptic stiffness around a nominal trajectory. This can be problematic in off-nominal situations, especially in regions of low variance, when the teleoperator attempts to perform actions that diverge from the nominal trajectory, e.g. due to an unplanned obstacle or out-of-sequence work. In these situations, the haptic stiffness actually impedes the ability of the operator to accomplish an objective since it is attempting to keep the operator around the nominal condition. In Chapter 4, this thesis primarily builds on two methods for handling off-nominal behavior.

In [**?**], magnitude of human deviation from an optimal path (in Zermelo's navigation problem) is used to indicate off-nominal situations and subsequently reduce the amount of assistance in shared control. This thesis draws from this concept but fundamentally differs in two ways: (i) imitation learning is used to produce a nominal assistive feedforward force since the optimal path is not analytically available; and (ii) human actions are classified as nominal or off-nominal using the expected forces exerted by the operator rather than the velocities from the model prediction.

Similarly in [**?**], magnitude of human-robot interaction forces in series-elastic actuators is used to adapt trajectory tracking compliance. In this work, transitions to a compliant state occur when the magnitude of forces exceeds a threshold. This thesis builds on this concept by (i) using predicted operator forces and variances to inform an adaptive transition to the compliant state (rather than select a heuristic threshold), and (ii) incorporates a hidden binary variable and transition model to leverage Bayesian estimation as in [**?**] for inferring the underlying nominal / off-nominal conditions.

2.4 Policies from imitation learning

This thesis uses imitation learning (alternately "learning from demonstration") [**?**, **?**] to develop models of human behavior and autonomous policies from expert demonstrations. There are two general approaches to the imitation learning problem: *behavior cloning* and *apprenticeship learning*.

Behavior cloning

Behavior cloning develops a direct function mapping from state X to action U by the policy: $\pi : X \to U$. This constitutes a supervised learning problem, with regression in the context of continuous action spaces U, or classification in discrete U [**?**]. Different tasks require different model realizations of π, resulting in several approaches to behavior cloning.

Temporal trajectories. Time-based methods [**?**, **?**, **?**, **?**] produce trajectories as a function of time, i.e. $T \to X$, and are well-suited to precisely automating repetitive tasks. However, these methods are rigid with respect to initial conditions and can have trouble with stochastic temporal data [**?**, **?**, **?**], making them ill-suited for providing active assistance in shared control or predicting noisy human actions. This thesis uses time-based trajectories to automate a cleaning task in Chapter 5, however, because the demonstration data is stochastic in time, the trajectories are found via optimization over dynamical system maps of demonstrated motion from selected initial conditions, rather than attempting to imitate the demonstrated trajectories in time directly.

Dynamical systems. Dynamical systems (DS) methods [**?**, **?**, **?**, **?**] produce models that predict time-rate of change in state, i.e. $X \to \dot{X}$, making them capable of motion generation and prediction from different initial conditions since the time evolution is implicit in the model output. However, these methods average over any explicit time-dependencies in the motion generation, and assume a well-defined unique target state for each model, making them primarily suited for reaching motions. This target state is often an equilibrium in the control sense, i.e. when velocity and acceleration are zero, i.e. $\dot{X} = 0$. This assumption is useful because of its stability implications and natural division of complex tasks into primitive motions that can be demonstrated independently. However, such equilibria do not exist in demonstrations of the example manufacturing task examined in Chapter 5. To address this issue, this thesis proposes a procedure to identify candidate target states that need not be state equilibria. Through this identification, motion can be decomposed into a

set of primitives, to which dynamical system methods can be applied.

Dynamic movement primitives (DMP) [**?**, **?**] produce a stable nonlinear attractor around the target state, supporting online adaptation to moving targets. Learning a DMP from demonstration involves learning feedback gains and a nonlinear forcing function parameterized by Gaussian kernels. DMPs also support learning rhythmic movements by choice of a different forcing function structure. However, DMPs model variables independently, leading to poor models in correlated data [**?**]. If support for probabilistic models is needed, e.g. for computing confidence intervals in the control policy, probabilistic movement primitives (ProMPs) [**?**] have been proposed as a probabilistic generalization to DMPs.

This thesis uses Gaussian mixture regression (GMR) [**?**] to model dynamical systems. GMR is a generalization to DMPs that additionally includes variable correlations. GMR produces locally linear models, making them suited to feedback stabilization around the target position using PD gain selection [**?**] or the SEDS technique [**?**]. Typically, GMR is used to produce first-order dynamical systems (i.e. predict velocity from position) [**?**,**?**,**?**], however this thesis proposes a second-order dynamical system to explicitly model accelerations during the cleaning task motion, which to the author's knowledge is the first such case using GMR.

To generalize GMR to dynamical systems with a time-depedency, i.e. $T \times X \rightarrow \dot{X}$, hidden Markov models (HMM) have been applied to model transitions between local clusters in time [**?**, **?**]. However, one challenge with this approach is how to choose the number of hidden states in the HMM, since this buffer influences the impact of previous clusters on the next cluster in the sequence. In the manufacturing task in Chapter 5, this choice is unclear. For example, in a manufacturing task, a sequence of repetitive motions may occur before a different final motion diverges from the repetitive motion in order to complete the task. To address this issue, this thesis proposes the use of a nonlinear autoregressive exogeneous (NARX) recurrent neural networks (RNN) [**?**] instead of HMM, where the exogeneous input is an accumulation feature, e.g. mechanical work, that indicates how much of the task has been performed.

Learning impedance and force controllers. A subset of imitation learning research concerns learning an impedance controller [**?**] that replicates both the motion and impedance characteristics of the demonstration [**?**, **?**, **?**], i.e. for interaction control. This holds relevance for manufacturing tasks involving contact with physical constraints (e.g. physical assembly operations), especially when the accuracy of the system is less than the positioning requirement of the physical task [**?**, **?**, **?**, **?**]. These methods are also used to address safety aspects of human-robot interaction.

This thesis employs a basic impedance control learning approach in Chapter 4 to develop an impedance control policy for reaching motions. Demonstrations are assumed to be compliant, i.e. all trajectories converge to a final target state. The policy is used to provide assistance in haptic shared control, however this approach is not suited for task automation, since the learning algorithm assumes unimodality and averages over complex multimodal accelerations and interaction forces that exist in the manufacturing task.

More rigorous approaches can be employed for automation. The unified motion and impedance controller (UMIC) [**?**] uses potential fields to represent a generalized impedance controller around a dynamical system, similar to SEDS [**?**]. Limitations of this method are that (i) potential fields are applied to each demonstration data point, causing the controller to be attracted to previous trajectories; and (ii) stiffness characteristics of the demonstration are assumed to be provided with the training data.

The GMR/HMM framework from [**?**] has been extended to the force control domain [**?**] for a door opening task. This approach encodes stiffness dynamics into the controller, as is therefore suited to controlling grasping stiffness. However, in manufacturing tasks such as cleaning, interaction forces are predominantly generated by friction, therefore Chapter 5 of this thesis proposes the use of second-order dynamics to predict interaction forces.

Apprenticeship learning

Although this thesis does not employ apprenticeship learning, aspects of the methods presented in Chapter 5 are related to apprenticeship learning, and it is discussed as a future

work in Chapter 7, therefore the approach is briefly discussed here.

Apprenticeship learning [**?**] assumes the operator chooses actions in similar fashion to a Markov Decision Process (MDP) [**?**, **?**] or optimal controller [**?**], i.e. by optimizing for some objective over dynamics. Unlike behavior cloning, apprenticeship learning produces a succinct reward function $R(X, U)$ (alt. a cost function), that generalizes to new dynamics. Learning a reward function is achieved with inverse reinforcement learning (IRL), alt. inverse optimal control (IOC) [**?**, **?**, **?**, **?**, **?**, **?**].

A challenge with inverse reinforcement learning is the ambiguity of the reward function, i.e. different reward functions could similarly describe a signel motion. One well-known approach to address this issue is feature expectation matching [**?**] and its resulting variants, where the reward function $R(X, U) = \theta^\top \phi(X, U)$ is a linear combination of a library of features $\phi(X, U)$, and the learning method produces estimates of the weights θ. With this approach, the number of demonstrations required is proportional to the number of features. Examples of features include distance to objects, total time, and magnitude of effort.

2.4.1 Summary

Based on the literature survey, dynamic shared autonomy shows promise towards reaching the project objectives of reducing task completion time and operator workload in limited-access manufacturing. However, practical challenges in localization, off-nominal detection, and task automation make realizing shared autonomy difficult without modification to existing methods. The next three chapters will develop aspects for a new approach, starting with addressing human-robot disagreement in traded control due to visual localization errors in Chapter 3; using interaction forces to detect off-nominal situations in Chapter 4; and culminating with the main contribution–learning a task dynamics model to address these limitations and automate the task–in Chapter 5.

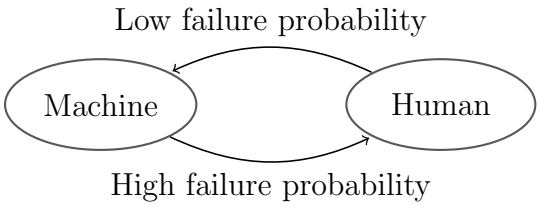
Chapter 3

CONTROL TRADING BASED ON LOCALIZATION CONFIDENCE

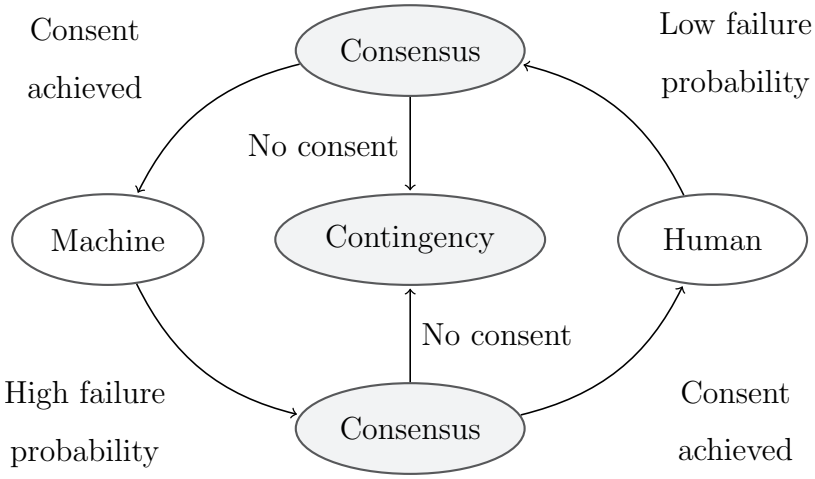
This chapter discusses traded control from confidence in localization estimates. As discussed in Chapter 1, uncertain placement of the robot arm in a remote location requires the need to localize the robot to the work. Due to time and workspace contraints, it is challenging to calibrate the robot position to the work before operation or deploy sensor arrays into the workspace. When target locations are not known within a reasonable degree, then the human is needed to perform the task, during which the robot can improve a localization estimate using feedback data from sensors onboard the robot. This chapter is published in the proceedings of the IEEE Conference on Advanced Intelligent Mechatronics (AIM 2017) [**?**].

To facilitate this type of shared autonomy, there is a need for trading control initiative [**?**] between fully automated (machine) mode and teleoperated (human) mode in robotic confined-space assembly of aircraft as shown in Fig. 3.1 (a). In mixed initiative interactions, both the human and machine agents can take control and perform the next action. [**?**]. This chapter examines the use of mixed-initiative traded control [**?**, **?**, **?**, **?**] between automation and teleoperation based on confidence in the localization estimate.

A challenge when both the human and the machine can propose a control trade is the potential for conflict when the human and machine do not agree on who is best suited for a given task as in Fig. 3.2, even when agreement is sought through communication as in [**?**, **?**]. Typically such conflicts in the mixed-initiative approach are addressed by giving ultimate authority to either the human or the machine. For example, in some mobile search and rescue applications [**?**, **?**], ultimate authority resides with the human. Conversely, in other mobile robot search applications, the robot retains veto power if it deems a human command



(a) Mixed-initiative traded control



(b) Mixed-initiative traded control with proposed consensus procedure

Figure 3.1: (a) In traded control, the machine or human agent has exclusive control of a system at any point in time. Mixed-initiative trades in control can be proposed by either the machine or human based on agent-specific models of failure probability. Disagreement stems from differences in the agents' models of failure, and occurs when the agents do not agree to a proposed trade. (b) The consensus-based approach introduced in this chapter addresses cases of disagreement by either (i) completing the control trade if the agent who needs to take over the control initiative consents, or (ii) enacting a contingency if consent is not given.

to be potentially dangerous [**?**]. In some cases the robot might not be able to continue the operation under high probability of failure, but the human might not be ready to assume

control.

To address this challenge, this chapter proposes an explicit consensus procedure with contingency procedures for the case when agreement on the proposed control trade is not achieved, i.e., a disagreement, as illustrated in Fig. 3.1 (b). Either agent (human $\mathcal{H}$ or machine $\mathcal{M}$) can propose a trade in the control initiative. If the other agent (which did not propose the trade) consents, then the trade is completed through a handoff procedure. However, if consent is not given, then a task-specific contingency procedure is enacted.

The chapter begins by discussing the novel consensus-based traded control scheme, including contingency procedures for resolving conflict in initiative. Next, particle filter localization using computer vision features, derivation of confidence estimates for a peg-in-hole geometry, and design of the task-specific consensus-procedures are described. An extension of the technique to multi-agent problems is presented, illustrating the generality of the contribution. The chapter concludes with experimental results illustrating improvements in completion time using the traded-control technique as compared with teleoperation.

3.1 Consensus-Based Traded Control

In mixed-initiative interactions, both agents (human and machine) can take the initiative and perform the next action. When an agent takes the initiative, the control trade is completed whether or not the other agent wishes to give up control.

3.1.1 Consensus-based mixed-initiative trade proposals

In the consensus-based mixed-initiative approach introduced in this chapter, both agents (human and machine) can propose trades in control initiative, but the ultimate decision on whether to perform the trade is determined through a consensus development procedure.

Definition 1 Proposal. *A proposal is a suggestion made by the proposing agent to the non-proposing agent for a control trade. The two types of control trade proposals are: i) a proposal to acquire control, or ii) a proposal to relinquish control.*

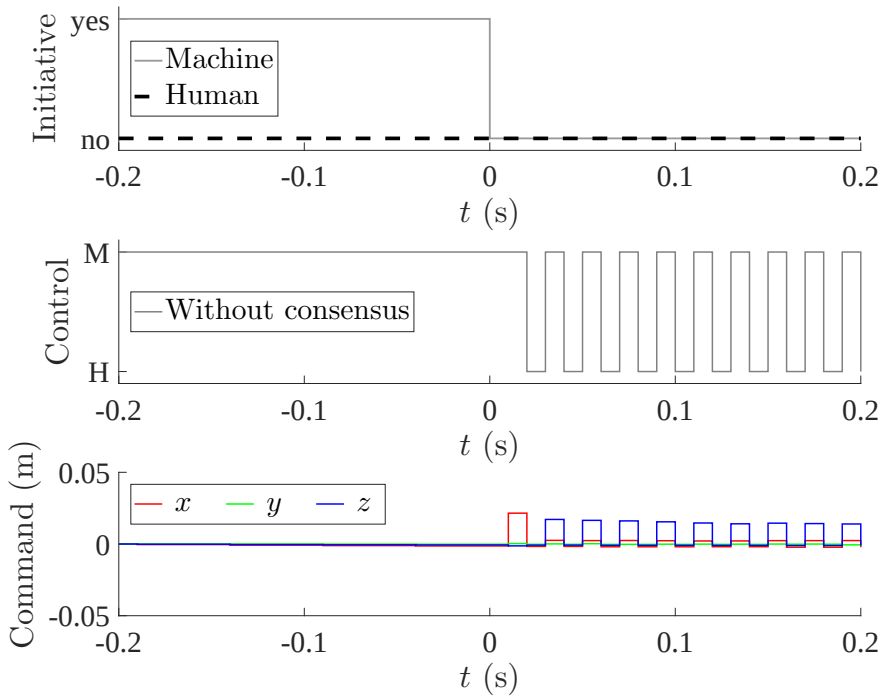


Figure 3.2: In mixed-initiative traded control, agent disagreement can arise when both agents propose to relinquish control, as shown at time $t = 0$ (both have no initiative). If no procedure is defined to address this issue, the disagreement results in a high-frequency limit cycle that switches control between the human ($\mathcal{H}$) and machine ($\mathcal{M}$) leading to a high-frequency limit cycle in the robot command (relative tool positions x, y, z). This paper introduces consensus-development procedures to explicitly address this agent disagreement.

The human can either propose to acquire control (if the machine currently has control) or propose to relinquish control (if the human currently has control). In instances where human intent cannot be directly sensed, it must be inferred from available data. The machine can also propose trades in control. Consider a machine that estimates the probability of failure $\hat{P}$ for a task. The machine can propose trades by comparing the probability of task failure $\hat{P}$ to a critical threshold P^*. If the estimated failure probability is high ($\hat{P} > P^*$) under machine control, then the machine proposes to relinquish control. If the estimated failure probability

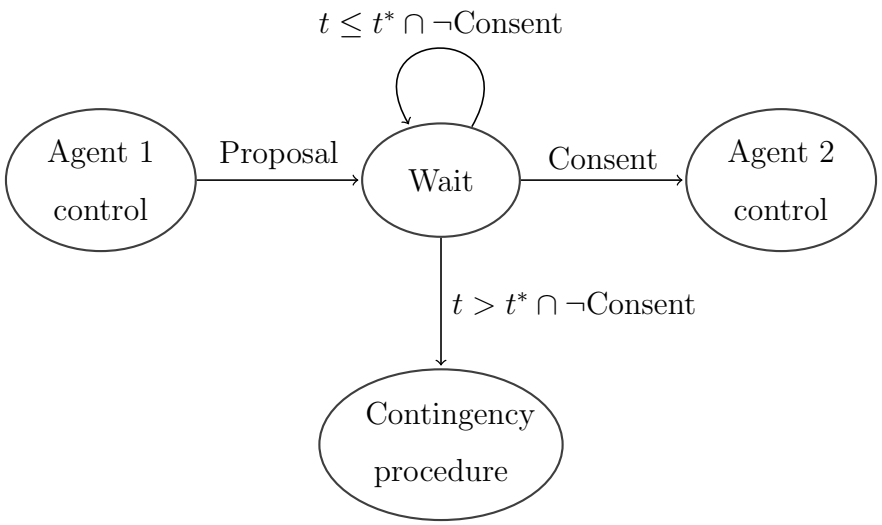


Figure 3.3: The consensus procedure begins when an agent proposes a trade in control. The procedure enters a wait state to provide the non-proposing agent sufficient time to gather situational awareness. If the non-proposing agent consents to the trade within the timeout period t^*, the trade occurs. However, if the agent does not consent within the timeout period, a disagreement has occurred and a contingency procedure must be enacted.

is low ($\hat{P} \leq P^*$) under human control, then the machine proposes to acquire control. The implementation of the proposed approach requires estimation of the probability of failure $\hat{P}$ from sensor data.

3.1.2 Procedure for consensus

Previous works address conflict in agent initiative by giving up control to the human [**?**, **?**] or by providing veto power to the machine [**?**, **?**]. Applications such as confined space manufacturing however have a variety of situations where the correct action is asymmetric, i.e. specific to the agent in control and agent proposing the control trade. The consensus development procedure shown in Fig. 3.3 accommodates potential disagreements in the control trade by using wait states and contingency procedures, as discussed below.

Wait state

While the consensus process is being developed, a timeout state can allow the potentially-consenting agent sufficient time to gain situational awareness.

Definition 2 Consent. *Consent is permission—based on specific criteria—granted by the non-proposing agent to execute a control trade proposal.*

If the non-proposing agent consents within a time period t^*, then the control trade occurs, otherwise consensus is not achieved. The timeout threshold t^* should be specific to the consenting agent, i.e. the timeout threshold $t^* = t^*_{\mathcal{M}}$ for trades requiring machine consent can be different from the timeout threshold $t^* = t^*_{\mathcal{H}}$ for trades requiring human consent. In particular, the machine timeout threshold $t^*_{\mathcal{M}}$ should be sufficiently large to allow the automation to complete computations and any (re)planning of routines based on computing hardware capabilities. The human timeout threshold $t^*_{\mathcal{H}}$ should be sufficiently large for an operator to gain situational awareness.

Contingency procedure

The final issue to consider in the consensus-based traded control is when the non-proposing agent does not consent to a trade.

Definition 3 Disagreement. *If consensus is not reached within the timeout period, the agents are in disagreement over the control trade.*

If agent disagreement occurs, then a contingency must be enacted to resolve the disagreement and avoid limit cycling of the control initiative. For many applications, the goal of the contingency might be to avoid a failure and/or keep personnel or hardware safe. The contingency could also provide a lockout operation, where an operator must check the machine status prior to operation resuming. Examples of contingencies are the machine holds position, the machine returns to a home operational state, the trade does not occur (i.e. control initiative remains with the original agent), or the trade is forced to occur (i.e. control trades

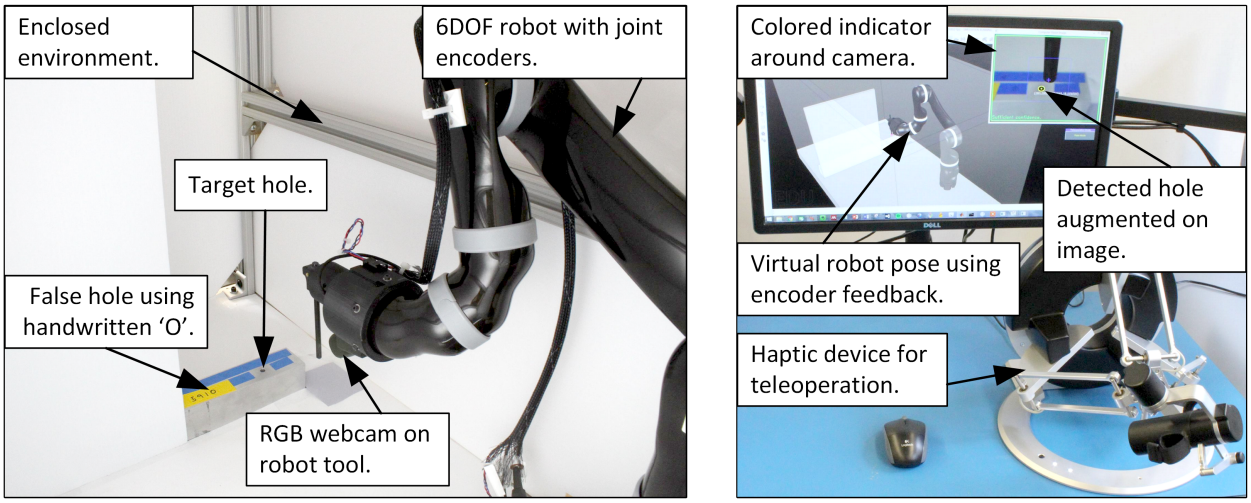
to the new agent). Such forced trading can still avoid limit cycling of the control initiative if it is restricted to one direction, e.g., from machine to human.

3.2 System Description

3.2.1 The task

The example task investigated here is the insertion of a peg into a target pilot hole $\boldsymbol{x}_t \in \mathbb{R}^3$ inside a confined space. This peg-in-hole insertion task captures the major challenges in fine positioning of a tool for assembly operations such as backdrilling inside an airplane wing. For example, the lack of direct visual access of the robot or the hole can make it challenging for a mechanic to teleoperate the robot from outside the confined space.

A camera (mounted on the robot close to the tool, i.e., the peg) is used to provide a first-person visual feedback of the tool (peg) and its surrounding area to the human operator. The operator is also provided with a virtual view of the robot pose inside the confined space using CAD data. This CAD information along with the view from the visual feedback from the camera is used by the operator to maneuver the robot inside the confined space for the peg insertion task. The robot mounted camera image feedback $\boldsymbol{z}_t$ is also used by the machine to estimate the target hole position $\boldsymbol{x}_t$. Camera limitations affect the ability of a machine controller to accurately estimate the target-hole location $\boldsymbol{x}_t$ and adapt the time trajectory of the robot position $\boldsymbol{x}_r$ to account for uncertainties in the target-hole location. In particular, the machine uses a particle filter to obtain an estimate $\mathrm{P}(\boldsymbol{x}_t)$ of the target hole location $\boldsymbol{x}_t$ and then adapts an initial nominal trajectory of the robot position $\boldsymbol{x}_r$ for completing the peg insertion task. Nevertheless, failures, such as collisions of the peg with the structure in the presence of large uncertainty in the target hole location's estimate $\mathrm{P}(\boldsymbol{x}_t)$ can result in expensive, time consuming repairs. Therefore, the goal is to exploit the advantages of both the human operator and the machine controller to improve the performance in terms of i) task success rate and ii) task completion time.



(a) Robot in enclosed environment.

(b) Remote operator workstation.

Figure 3.4: The platform (a) consists of a robot inserting a peg into a pilot hole in a confined space using computer vision feedback. The operator remotely interfaces with the system (b) using a haptic device for teleoperation and is provided a virtual view of the robot in the confined space and real-time camera feedback augmented with computer vision hole detection.

3.2.2 Machine failure probability estimation

An uncertain target hole location $\boldsymbol{x}_t$ results in a primary automation failure mode due to collision $\mathcal{C}$ of the robot peg with material surrounding the target hole. The machine trigger $I_{\mathcal{M}}$ is determined by comparing the estimated probability of failure $\hat{P}$ to the threshold P^*

$$I_{\mathcal{M}} = \begin{cases} \text{true} & \text{for } \hat{P} \leq P^* \\ \text{false} & \text{for } \hat{P} > P^*. \end{cases} \tag{3.1}$$

When the machine initiative $I_{\mathcal{M}}$ is true, the machine will propose a trade to machine control or consent to such a trade. When the initiative $I_{\mathcal{M}}$ is false, the machine will propose a trade to human control or consent to such a trade.

There are two aspects to estimating the probability of failure $\hat{P} = P(\mathcal{C})$, i.e. the probabil-

ity of collision between the peg and the coupon around the hole. These aspects are: (i) use of a particle filter for estimating the uncertainty in target hole location $\boldsymbol{x}_t$; and (ii) estimating the probability of failure from the uncertainty in the target hole location estimate $P(\mathcal{C}|\boldsymbol{x}_t)$.

Estimating uncertainty

Let $\boldsymbol{x}_t \in \mathbb{R}^3 \sim \mathcal{N}(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{\Sigma}}_t)$ be the target pilot hole location in the world coordinate frame with prior mean $\bar{\boldsymbol{x}}_t$ and covariance $\bar{\boldsymbol{\Sigma}}_t$.

Estimating the target-hole location. The target hole is detected by a standard definition (SD) monocular camera and projected onto the image plane as an ellipse. Grayscale ellipse detection implemented in OpenCV 2.4 filters candidate ellipses by compactness [**?**]. The algorithm returns an ellipse feature measurement vector

$$\boldsymbol{z}_t = (u,\ v,\ r)^\top \tag{3.2}$$

for the most compact remaining ellipse, where $(u,\ v)$ is the ellipse centroid in pixels and r is the length of the ellipse major axis in pixels. Note the target pilot hole radius is known a-priori. The algorithm returns $\boldsymbol{z}_t = \boldsymbol{0}$ if no ellipse is detected. Using a pinhole camera representation [**?**, **?**, **?**], the target-hole location in the camera frame $\boldsymbol{X}_t = (X,\ Y,\ Z)^\top$, is projected into ellipse features by perspective projection $\boldsymbol{z}_t = \boldsymbol{h}(\boldsymbol{x})$, defined as

$$\boldsymbol{z}_t = \boldsymbol{h}(\boldsymbol{X}_t) = \frac{1}{Z}\begin{pmatrix} Xf_u \\ Yf_v \\ f_r \end{pmatrix} + \begin{pmatrix} c_u \\ c_v \\ c_r \end{pmatrix} \tag{3.3}$$

where $(f_u,\ f_v,\ f_r,\ c_u,\ c_v,\ c_r)$ are intrinsic camera parameters determined from calibration. The camera pose is represented by a homogeneous coordinate transformation $\boldsymbol{T}(\boldsymbol{x}_r)$ that maps the target hole location $\boldsymbol{X}_t^*$ in the camera frame (* denotes homogeneous coordinates, i.e. $\boldsymbol{x}^* = (\boldsymbol{x}^\top,\ 1)^\top$) to its location in the world $\boldsymbol{x}_t^*$, i.e. $\boldsymbol{T} : \boldsymbol{X}_t^* \to \boldsymbol{x}_t^*$, defined by

$$\boldsymbol{T}(\boldsymbol{x}_r) = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t}(\boldsymbol{x}_r) \\ \boldsymbol{0}^\top & 1 \end{pmatrix} \tag{3.4}$$

where $\boldsymbol{R} \in \mathbb{SO}_3$ is a rotation matrix defined by extrinsic Euler angles and $\boldsymbol{t} \in \mathbb{R}^3$ is a translation vector defined by robot position $\boldsymbol{x}_r$ and extrinsic camera parameter offsets. Extrinsic parameters are computed by an extrinsic calibration routine. The expected value of the target hole observation is then

$$\mathbb{E}[\boldsymbol{z}_t] = h(\boldsymbol{T}(\boldsymbol{x}_r)^{-1}\boldsymbol{x}_t^*). \tag{3.5}$$

Particle filter for estimating the target posterior density. A recursive Bayesian filter framework [**?**] with sample index $k \in \mathbb{I}^+$, approximated by a particle filter [**?**, **?**], is used to model the posterior density of the target hole location $\boldsymbol{x}_t$ from the feature observation $\boldsymbol{z}_t$. The prediction step

$$\mathrm{Bel}(\boldsymbol{x}_t) = \int \mathrm{P}(\boldsymbol{x}_t^k \mid \boldsymbol{x}_t^{k-1})\, \mathrm{P}(\boldsymbol{x}_t^{k-1})\; d\boldsymbol{x}_t^{k-1} \tag{3.6}$$

incorporates a prediction model $\mathrm{P}(\boldsymbol{x}_t^k \mid \boldsymbol{x}_t^{k-1}) = \mathcal{N}(\boldsymbol{x}_t^{k-1}, \boldsymbol{Q})$ with process covariance $\boldsymbol{Q} \in \mathbb{R}^{3\times3}$. This prediction models the target hole position as motionless in the world coordinate frame. Covariance $\boldsymbol{Q}$ is a contrived uncertainty that deflates the belief in the hole prediction model, with diagonal elements chosen to be $\sigma^2 = 5 \cdot 10^{-7}$, resulting in a model standard deviation of ~ 0.7 mm. This is added to ensure the particle filter samples from a sufficient proposal region to avoid overconfidence and particle collapse. The prior target hole density is initialized to $\mathrm{P}(\boldsymbol{x}_t^0) = \mathcal{N}(\bar{\boldsymbol{x}}_t, \bar{\boldsymbol{\Sigma}}_t)$ for the first sample index $k = 1$.

The update step corrects the prediction from (3.6) with a camera feature measurement $\boldsymbol{z}_t$ according to

$$\mathrm{P}(\boldsymbol{x}_t^k \mid \boldsymbol{z}_t^k) \propto \mathrm{P}(\boldsymbol{z}_t^k \mid \boldsymbol{x}_t^k)\mathrm{Bel}(\boldsymbol{x}_t). \tag{3.7}$$

The likelihood $\mathrm{P}(\boldsymbol{z}_t \mid \boldsymbol{x}_t)$ in the update (3.7) is modeled as a mixture of likelihoods [**?**]

$$\mathrm{P}(\boldsymbol{z}_t \mid \boldsymbol{x}_t) = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}^\top \begin{pmatrix} \mathrm{P}_{\mathrm{TP}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t) \\ \mathrm{P}_{\mathrm{FP}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t) \\ \mathrm{P}_{\mathrm{FN}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t) \end{pmatrix} \tag{3.8}$$

where mixture weights θ_i are learned from data using Expectation-Maximization (EM) [**?**], and $\sum \theta_i = 1$. The densities are defined as follows:

1. $\mathrm{P_{TP}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t)$ models the likelihood of a true positive detection by the distribution

$$\mathrm{P_{TP}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t) = \begin{cases} \eta\mathcal{N}(\boldsymbol{z}_t \mid \mathbb{E}[\boldsymbol{z}_t], \boldsymbol{R}) & \text{if } \boldsymbol{z}_t \in \mathcal{D}_I \\ \mathcal{I}(\boldsymbol{z}_t = \mathbf{0}) & \text{else} \end{cases} \tag{3.9}$$

   where $\mathbb{E}[\boldsymbol{z}_t]$ is computed in (3.5), $\boldsymbol{R}$ is the measurement covariance found from test data, $\mathcal{D}_I$ is the image domain over which measurement features $\boldsymbol{z}_t$ are defined, $\mathcal{I}(\boldsymbol{z}_t = \mathbf{0})$ is a point-mass distribution centered on $\boldsymbol{z}_t = \mathbf{0}$, and η is a normalizing coefficient such that $\eta = \left(\int_{\mathcal{D}_I} \mathcal{N}(\cdot) d\boldsymbol{z}_t\right)^{-1}$;

2. $\mathrm{P_{FP}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t)$ models the likelihood of a false positive detection by the uniform distribution $\sim \mathcal{U}(\boldsymbol{z}_t \in \mathcal{D}_I)$ defined on the image domain $\mathcal{D}_I$; and

3. $\mathrm{P_{FN}}(\boldsymbol{z}_t \mid \boldsymbol{x}_t)$ models the likelihood of a false negative detection by the point-mass distribution $\mathcal{I}(\boldsymbol{z}_t = \mathbf{0})$.

A generic particle filter [**?**] approximates the Bayes filter in (3.6) and (3.7). The filter is initialized with 500 particles uniformly-distributed over the $\pm 6\sigma$ region of $\mathcal{N}(\bar{\boldsymbol{x}}_{\boldsymbol{t}}, \bar{\boldsymbol{\Sigma}}_{\boldsymbol{t}})$ with uniform weights. Filter updates are performed at 20Hz due to bandwidth limitations of the monocular camera, and systematic resampling is performed when the number of effective particles $\hat{N}_{eff}$ [**?**] is less than 100. The particle filter returns a target hole location estimate $\hat{\boldsymbol{x}}_t$ and covariance estimate $\hat{\boldsymbol{\Sigma}}_t$ computed using particle weights. This is sufficient since the posterior density is expected to be unimodal and resultant operations on the estimate assume a Gaussian form.

Estimating probability of collision

The collision probability model $\mathrm{P}(\mathcal{C} \mid \boldsymbol{x}_r, \hat{\boldsymbol{x}}_t, \hat{\boldsymbol{\Sigma}}_t)$ is conditioned on robot position $\boldsymbol{x}_r$ and target hole position statistics $\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{\Sigma}}_t$ computed from the particle filter. For simplicity, the model only considers collision with the work piece material near to the target hole.

Collision regions. Consider the vertical condition in Fig. 3.5 (a). A collision occurs when the vertical position of the robot peg z_r is below the target hole plane at z_t, i.e.,

$$\psi_1 : \quad z_r \leq z_t. \tag{3.10}$$

When ψ_1 is met, and the distance between the centerline of the robot peg (x_r, y_r) and the target hole centerline (x_t, y_t) exceeds the difference in peg and hole radii $r_p - r_h$, i.e.

$$\psi_2 : \quad (r_h - r_p)^2 \leq (x_r - x_t)^2 + (y_r - y_t)^2, \tag{3.11}$$

then a collision occurs, as in in Fig. 3.5 (b). Thus the probability of collision is the intersection of the probability of satisfying both ψ_1 and ψ_2 constraints, according to

$$\mathrm{P}(\mathcal{C} \mid \boldsymbol{x}_r, \hat{\boldsymbol{x}}_t, \hat{\boldsymbol{\Sigma}}_t) = P(\psi_1 \cap \psi_2) \tag{3.12}$$

$$= \int_{\psi_1 \cap \psi_2} \mathrm{P}(\boldsymbol{x}_t)\ d\boldsymbol{x}_t. \tag{3.13}$$

Note that $\mathrm{P}(\psi_1)$ and $\mathrm{P}(\psi_2)$ are independent if there is no correlation in the covariance between $\vec{z}$ and $\vec{x}$ or $\vec{y}$. This is not the case for this application, therefore a numerical approach to solve (3.13) is needed.

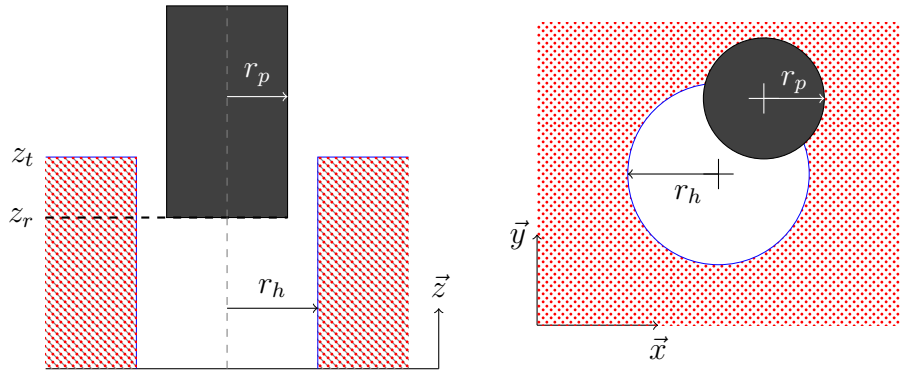
Numerical solution. Let $\mathbb{I}(\boldsymbol{x}_t)$ be a binary indicator to denote whether $\boldsymbol{x}_t$ is in a collision region in space (i.e. a binary occupancy map), defined as

$$\mathbb{I}(\boldsymbol{x}_t) = \begin{cases} 1 & \boldsymbol{x}_t \in (\psi_1 \cap \psi_2) \\ 0 & \text{else} \end{cases} \tag{3.14}$$

Because the particle filter is approximating the posterior density of the target hole $\mathrm{P}(\boldsymbol{x}_t)$, (3.13) can be approximated by

$$\mathrm{P}(\mathcal{C}) \approxeq \sum_{i=1}^{N} w^{(i)} \mathbb{I}(\boldsymbol{x}_t^{(i)}) \tag{3.15}$$

where $(\boldsymbol{x}_t^{(i)},\ w^{(i)})$ is the target hypothesis and weight, respectively, for the i^{th} particle in the particle filter. Using this approach, the probability of collision $\mathrm{P}(\mathcal{C})$ can be computed for an arbitrary density $\mathrm{P}(\boldsymbol{x}_t)$ represented by the particle filter hypotheses and posterior density weights.



(a) Vertical view (b) Planar view

Figure 3.5: A collision of the peg with the area around the target hole occurs when a) the vertical position of the robot peg z_r is below the target hole plane z_t and b) the distance between the robot peg centerline (x_r, y_r) and the target hole centerline (x_t, y_t) exceeds the difference in the peg and hole radii, r_p and r_h, respectively.

3.2.3 Platform

The application uses a Kinova MICO 6DOF manipulator to insert a peg fixed to the end of the robot into the target pilot hole as shown in Fig. 3.4 (a). The Cartesian motion of the MICO is limited to a maximum absolute velocity of 4 cm/s in each axis, and the robot tool maintains a constant orientation with respect to the environment. A standard definition (SD) Logitech C270 webcam, attached to the robot, provides visual feedback of the area around the peg. The confined space is flooded with diffuse light via a Genaray LED-6800, and the computer vision threshold parameter is tuned a-priori to detect the target hole under maximum intensity (nominal) light. Successful insertion of the peg into the target hole is registered via a contact switch in the hole, and an unsuccessful test is registered when the peg is subjected to a force $>$ 10N in any direction, measured through a force/torque sensor on the robot tool. Real-time control and state estimation as well as the I/O and test logic were implemented on a Speedgoat Real-Time performance computer running compiled

MATLAB/Simulink software.

The operator interface is shown in Fig. 3.4 (b). Unilateral teleoperation is accomplished through a Force Dimension Omega.7 haptic device using virtual stiffness and damping to keep the tool centered around a nominal position. During runtime, the operator can use the computer mouse to toggle between rate or position control modes [**?**]. The operator requests control initiative $I_{\mathcal{H}}$ by pressing the gripper button on the Omega.7. An unpressed button indicates the human does not want the initiative, i.e. $\neg I_{\mathcal{H}}$. Remote visualization of the robot in the confined space from CAD data is accomplished using V-REP software.

3.2.4 Consensus procedures and contingencies

The machine timeout threshold is chosen to be twice the time step of the vision update, i.e. $t^*_{\mathcal{M}} = 0.1$s. The timeout for human response has a lower bound of the human response time of 100 ms [**?**], but in order to give the human sufficient time for reasoning, the timeout is chosen to be $t^*_{\mathcal{H}} = 1.0$s, an order of magnitude greater than the lower bound.

Table 3.1 outlines the four possible cases in the consensus development procedure and corresponding application-specific contingencies, which are discussed in detail below. Note that human initiative $I_{\mathcal{H}}$ and machine initiative $I_{\mathcal{M}}$ are used to represent proposed trades in control and consent to trades in control interchangeably, e.g.: if the machine initiative $I_{\mathcal{M}}$ is true, then the machine will either propose trades or consent to trades that provide the machine with control. If the machine initiative $I_{\mathcal{M}}$ is false however, the machine will either propose trades or consent to trades that provide the human with control.

Human proposes to acquire control

In this state, the machine currently has control and the human is attempting to take control. For the trade to occur, the machine must consent to relinquish control, indicated by $\neg I_{\mathcal{M}}$. If machine consent $\neg I_{\mathcal{M}}$ does not occur within the timeout period $t^*_{\mathcal{M}}$ from the initial time the human proposed the control trade, then the contingency must be enacted. For this

Table 3.1: Consensus procedures and contingencies.

	Proposal	**Consent**	t^*	**Contingency**
1)	$\mathcal{H}$ to acquire	$\neg I_{\mathcal{M}}$	$t^*_{\mathcal{M}}$	Human control
2)	$\mathcal{M}$ to relinquish	$I_{\mathcal{H}}$	$t^*_{\mathcal{H}}$	Robot hold pose
3)	$\mathcal{H}$ to relinquish	$I_{\mathcal{M}}$	$t^*_{\mathcal{M}}$	Human control
4)	$\mathcal{M}$ to acquire	$\neg I_{\mathcal{H}}$	$t^*_{\mathcal{H}}$	Human control

application, the contingency is to allow the trade to human control to occur, giving the human final authority for this transition path.

Machine proposes to relinquish control

In this state, the machine currently has control and is attempting to hand control over to the human. For the trade to occur, the human must consent to take control, indicated by $I_{\mathcal{H}}$. If human consent $I_{\mathcal{H}}$ does not occur within the timeout period $t^*_{\mathcal{H}}$ from the initial time the machine proposed the control trade, then the contingency must be enacted. For this application, the contingency is to leave control with the machine, but require the machine to hold its current position (i.e. maintain a zero velocity) until human consent $I_{\mathcal{H}}$ occurs. This also requires the machine to hold position during the *Wait* state while the human operator has not assumed control.

Human proposes to relinquish control

In this state, the human currently has control and is attempting to hand control over to the machine. For the trade to occur, the machine must consent to take control, indicated by $I_{\mathcal{M}}$. If machine consent $I_{\mathcal{M}}$ does not occur within the timeout period $t^*_{\mathcal{M}}$ from the initial time the human proposed the control trade, then the contingency must be enacted. In this

application, the contingency is for the human to retain control initiative, i.e. ignore the trade, due to the possibility that the machine may not be ready to take control.

Machine proposes to acquire control

In this state, the human currently has control and the machine is attempting to take control. For the trade to occur, the human must consent to relinquish control, indicated by $\neg I_{\mathcal{H}}$. If human consent $\neg I_{\mathcal{H}}$ does not occur within the timeout period $t^*_{\mathcal{H}}$ from the initial time the machine proposed the control trade, then the contingency must be enacted. For this application, the contingency is to leave the human with control due to the possibility that the human may not be ready to give up control.

3.3 Generalization to N Agents

The choices of contingency procedures in the previous section are motivated by the relative expertise of the two agents vying for control. Because the human has a more complete understanding of the task than the machine, the human is granted more oversight of the initiative in the contingencies. This section discusses the extension of this concept to systems with N agents (with possibly multiple human agents), provided each agent has a unique priority, i.e.,

$$\rho_i \neq \rho_j, \quad \forall\, i, j \in \{1, \ldots, N\} \text{ and } i \neq j, \tag{3.16}$$

where $\rho_i > \rho_j$ implies that agent i has higher priority than agent j. Dynamic priority assignment can be used to provide flexibility, i.e. one agent may have higher priority during one part of a task, and a lower priority during a second part of the task. Issues in switching of such priority-hierarchies is an area for future research.

3.3.1 Reduction to two agent disagreements

For systems with multiple N agents, the consensus development problem can be reduced to disagreements between two agents if at any given time only two agents are vying for control.

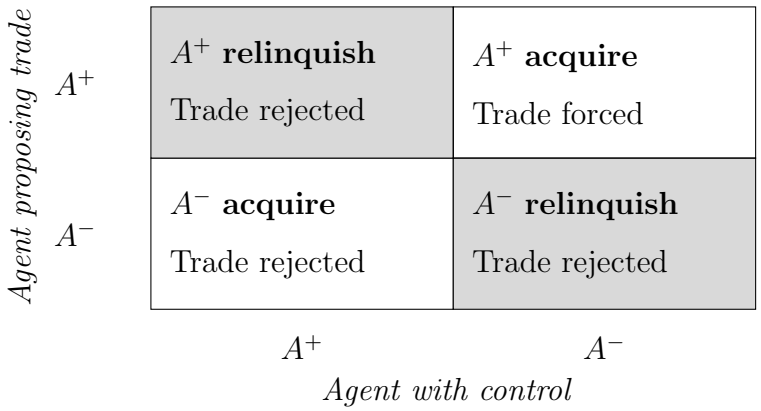


Figure 3.6: Generalized contingencies (when consent is not given) for a system comprised of a higher priority agent A^+ and a lower priority agent A^-. A trade is forced to occur only when the higher priority agent proposes to acquire control. In all other scenarios, the trade is rejected. Safety protocols in the contingency might be needed when the agent with the lower priority A^- wishes to relinquish control but the agent with the higher priorityA^+ does not accept.

In practice, if more than two agents are in disagreement, the problem can be reduced to two agents by first choosing the agent that has control and subsequently selecting the highest priority agent that has proposed a trade. Moreover, if the consensus development procedure is ongoing then a new trade could be entertained only after the current disagreement is resolved, which will limit disagreement to be between one pair of agents.

3.3.2 Contingencies based on agent priority

Re-examining Table 3.1, the machine $\mathcal{M}$ can be viewed as the agent with lower priority and the human as the agent with higher priority. Denote the agent with higher priority as A^+ and the agent with lower priority as A^-, i.e.:

$$\rho_{(A^+)} > \rho_{(A^-)}. \tag{3.17}$$

Then, generalized contingencies for two arbitrary agents is illustrated in Fig. 3.6. The contingency selection only forces a trade to occur when a higher priority agent A^+ is proposing to acquire control, i.e. attempting to take control from the lower priority agent A^-. This particular trade is important because the higher priority agent may have more advanced reasoning or more complete knowledge of the situation and could be attempting to stop the other agent from causing damage or harm.

The contingency selection prohibits the remaining trades from being forced. For example, if the lower priority agent A^- is proposing to relinquish control but the higher priority agent A^+ does not consent, then a contingency procedure to address safety might be needed if control cannot safely remain with the lower priority agent A^-. Under these conditions, in the current application, the lower priority agent (the machine) holds the current position, and thereby, avoids causing damage.

3.4 Experiment

A set of experiments evaluated i) autonomous operation, ii) teleoperation, and iii) mixed-initiative traded control approaches to the peg-in-hole assembly task where the objective is to successfully complete the task in minimum time. For traded control, the operator was instructed to let the machine try to complete as much of the task as possible to minimize completion time. 100 tasks were performed for each of the control modes i-iii) (totaling 300 task runs), and an error type I-III) was randomly selected from a uniform distribution for each task. The robot-perception error scenarios —designed to mimic situations in aerospace assembly operations and exercise consensus development in the mixed-initiative controller —are described below, with computer vision views for each case shown in Fig. 3.7.

- *Type I baseline error.* The initial target hole location estimate $\bar{\boldsymbol{x}}_t$ used to initialize the particle filter was injected with an additive error $\epsilon \sim \mathcal{N}(\mathbf{0}, \hat{\boldsymbol{\Sigma}}_t)$. The particle filter was designed to accommodate this level of error and thus provides a baseline for performance.

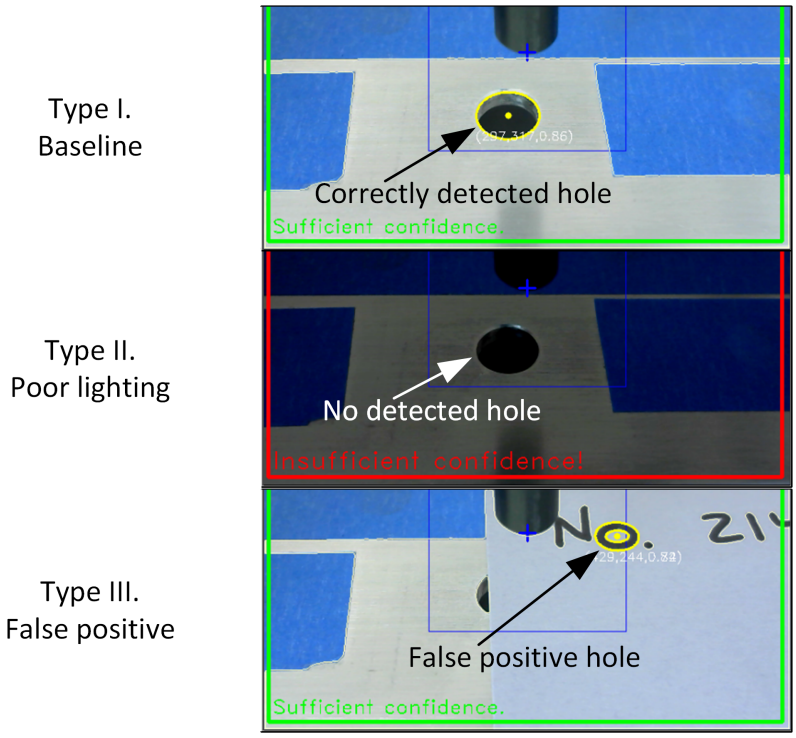


Figure 3.7: The system is exposed to three robot-perception error scenarios to evaluate the mixed-initiative traded controller with consensus-development.

- *Type II poor lighting error.* The initial target hole estimate $\bar{\boldsymbol{x}}_t$ was generated as in Type I error case. Additionally, the brightness and contrast parameters in the hole detection algorithm were varied sufficiently to yield a significantly higher false negative detection rate ($\tilde{5}$0-75%) from the baseline case ($\tilde{0}$-10%). This resulted in a decreased confidence in the target hole estimate $\hat{\boldsymbol{x}}_t$.

- *Type III false positive error.* The initial target hole $\bar{\boldsymbol{x}}_t$ was subjected to a gross error that placed the prior over a handwritten character 'O' (roughly the size of the pilot hole). Lighting was maintained to the baseline Type I error case. This error type resulted in a false positive detection and overconfidence in the handwritten 'O' being the target pilot hole.

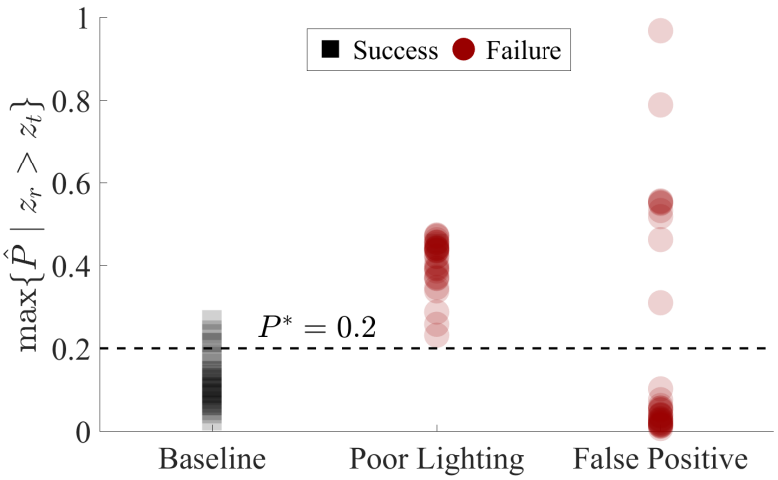


Figure 3.8: The machine initiative threshold P^* was chosen by comparing collision probability estimates $\hat{P}$ when the robot peg is above the target hole plane, i.e. $z_r > z_t$, under autonomous operation for error types I-III. Specifically, the maximum collision probability for Type II errors exceeded 0.2 for all runs, resulting in a choice of $P^* = 0.2$ for the machine to discern Type II errors from Type I and III.

Results for autonomous operation were used to identify the acceptable threshold parameter P^* for the traded control operation, as described below. The maximum computed probability of collision $\hat{P}$ prior to the plunge, i.e. when the robot peg is above the hole plane $z_r > z_t$, was recorded for each error type. The value of P^* that separated Type I errors from Type II errors was chosen to be the critical threshold for the mixed-initiative traded controller. Fig. 3.8 shows the separation in failure probability $\hat{P}$ used to determine machine control-initiative threshold P^*. Type II (poor lighting) errors result in a higher collision probability than Type I (baseline) errors, specifically with the collision (failure) probability $\hat{P} > 0.2$. Therefore, the threshold P^* was chosen to be 0.2. Type III (false positive) errors however cannot be separated from Type I (baseline) errors by this metric, as expected by the experiment design.

Table 3.2: Type I. Error (Baseline) Results

Method	**Runs**	**S** [%]	**T** ($\mu \pm \sigma$) [s]	**ANEES**
Autonomous	30	100	11.1±0.4	8.3
Teleoperation	28	82	23.4±5.0	-
Traded	37	100	11.7±1.6	-

3.5 Results and Discussion

Success rate S and task time T were evaluated for i) autonomous operation, ii) teleoperation, and iii) the mixed-initiative consensus-based traded controller for all 3 error types. Results are shown in Fig. 3.9 and presented in Tables 3.2, 3.3, and 3.4 by error type. Success rate S for Type II and III errors improved from 3% and 0%, respectively in pure automation to 97% and 100% with the traded controller. Additionally, mean completion time T for Type I errors was reduced from 23.4s in pure teleoperation to 11.7s with the traded controller.

Results from the study show the advantage of the traded controller over purely autonomous operation and pure teleoperation. The mixed-initiative traded controller uses the operator to increase success rates by over an order of magnitude with an accompanying increase in completion times (2× and 3×) for Type II (poor lighting) and III (false positive) errors, respectively. Additionally, the traded controller shows a 5% increase in completion time and no statistically significant impact (95% confidence) to success rate in Type I (baseline) errors.

The average normalized estimation error squared (ANEES) [**?**] credibility test values in Tables 3.2—3.4—defined by

$$\text{ANEES} = \frac{1}{nM}\sum_{i=1}^{M}(\boldsymbol{x}_{t,i} - \hat{\boldsymbol{x}}_{t,i})^{\top}\Sigma_i^{-1}(\boldsymbol{x}_{t,i} - \hat{\boldsymbol{x}}_{t,i}), \tag{3.18}$$

where $\boldsymbol{x}_{t,i}$ is the true hole position, $\hat{\boldsymbol{x}}_{t,i}$ is the estimated hole position, Σ_i is the estimated

Table 3.3: Type II. Error (Poor Lighting) Results

Method	**Runs**	**S** [%]	**T** $(\mu \pm \sigma)$ [**s**]	**ANEES**
Autonomous	33	3	10.7±0	4.5
Teleoperation	34	91	24.6±5.4	-
Traded	29	97	20.3±3.9	-

Table 3.4: Type III. Error (False Positive) Results

Method	**Runs**	**S** [%]	**T** $(\mu \pm \sigma)$ [**s**]	**ANEES**
Autonomous	37	0	NaN	2400
Teleoperation	38	89	23.9±5.1	-
Traded	34	100	26.2±4.1	-

covariance, and n is the number of samples for run $i \in \{1, \ldots, M\}$—provide insight into the influence of error type on the machine initiative. The ANEES over the Type I tests shows that the particle filter is optimistic under baseline operation (ANEES > 1 indicates overconfidence). However, the ANEES for the Type II error (false negative) is more pessimistic, leading to a higher estimated probability of collision $\hat{P}$. The higher collision probability, by design of the threshold P^*, yields a machine-proposed trade from machine to human control. The ANEES for the Type III false positive error (i.e., the error the perception cannot detect) indicates an estimator that is grossly overconfident. In this case, the human operator must request a trade in control because the perception algorithm is not able to detect the error.

Occurrence of consent requests for control transition and enactment of contingency procedures with the mixed-initiative traded controller is depicted in Fig. 3.10 for different error modes. With Type I (baseline) errors, the machine proposed to relinquish control to the

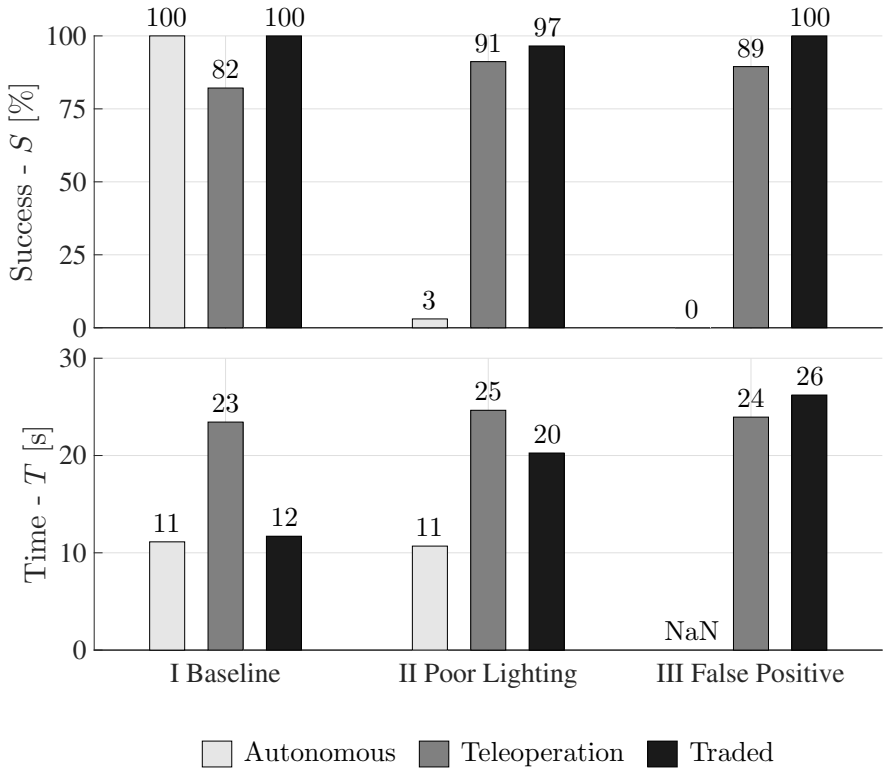


Figure 3.9: A performance evaluation of the traded controller for a total of 300 tests shows a decrease in task time T over pure teleoperation for Type I errors and an increase in success rate S over purely autonomous operation for Type II and III errors.

human in 25% of tests due to occasional detection loss, but the operator never took control within the specified time interval, resulting in a hold-pose contingency. If more data became available and the detection loss was overcome, then the machine would resume operations and successfully complete the insertion task. With Type II (poor lighting) errors, the machine relinquished control to the human 100% of the tests as expected. When the operator did not accept control within the timeout period (in >50% of the tests), the hold-pose contingency was enacted again. Finally, with Type III errors (false positive), the human had to request control from the machine 92% of the tests. However, perception overconfidence led to continued request to acquire control from the machine and the repeated enactment of

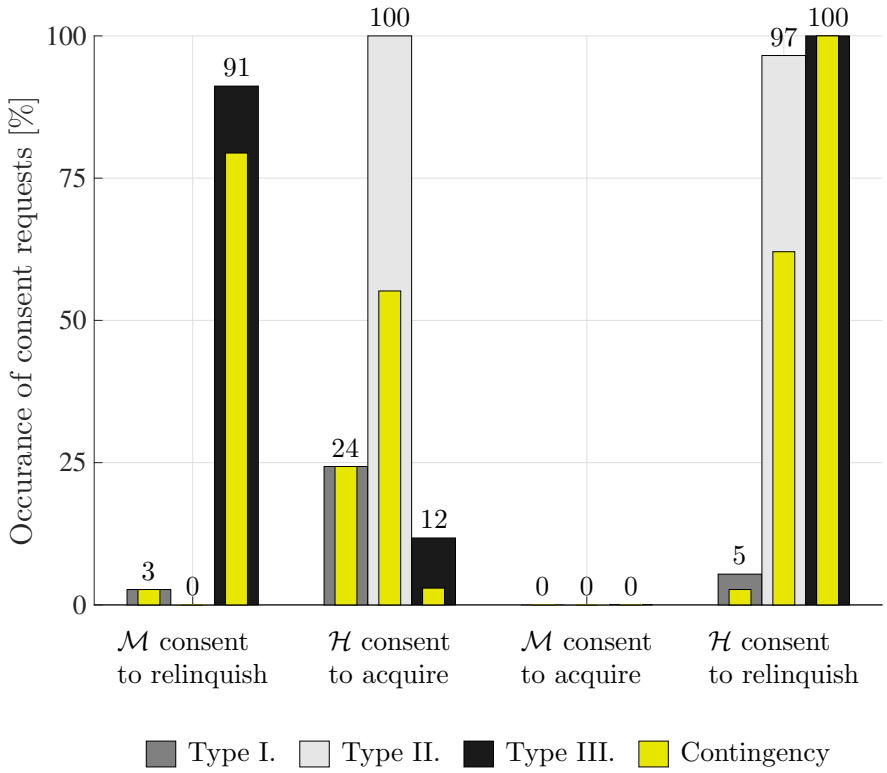


Figure 3.10: Occurrence of consent requests, as in Table 3.1, in trials under different error cases (Type I. baseline, Type II. poor lighting, and Type III. false positive). The relative occurrence of enacting the contingency procedure when consent is not obtained is shown in the yellow inner bar.

keep-human-control contingency.

Without a consensus-development procedure, the traded controller exhibits a limit cycle in situations of agent initiative conflict as shown in Fig. 3.2. The 50Hz frequency of this limit cycle is half of the 100Hz sample frequency of the traded controller, as expected. In this experiment, the traded controller was tested under a Type III (false positive) error condition with and without the consensus development procedure. When the consensus-development procedure introduced by this paper is used, the traded controller properly trades control to the human without limit cycling.

3.6 Conclusion

This chapter introduced mixed-initiative traded control, where control switches back and forth between human and machine based on agent initiatives. For the automation, this initiative was computed using the confidence in localization estimates with respect to the peg-in-hole geometry. In principal, any feedback method could be used for localization, requiring an update to the senor likelihood in (3.7). The main contribution of this chapter was the development of *consensus-based traded control*, where contingency procedures are enacted in the case of disagreement between the human and robot. Experimental results on a peg-in-hole task showed that the method enables traded control to (i) reduce completion time by 50% as compared with teleoperation, and (ii) increase task success rate from 3% in pure automation to 97% when localization failure modes are introduced.

There are several limitations with the methods presented in this chapter.

- Computation of the machine initiative relies on computing confidence in the localization estimate. When the estimate is underconfident (false negative) or uncertain (true negative), the initiative will be low and the machine will not accept control. When the estimate is confident and correct (true positive), the machine will propose to take control, in which case the remainder of the task can be automated. The problem arises when the estimator is overconfident and incorrect (false positive). With the traded control scheme presented in this chapter, the *human* is expected to intervene. However, if the human is not paying attention, the robot will continue with potential for damage. Therefore, this method alone is not sufficient for robust system operation.

- Another limitation that arises is the sensitivity of the vision system to lighting conditions. The experiments used controlled lighting. Unfortunately, large datasets of limited-access manufacturing operations are not available, and traditional CV techniques do not perform well for the low-lighting conditions that arise in practice. Therefore alternate or additional feedback data is needed for robust localization.

Chapter 4

MANAGING OFF-NOMINAL SITUATIONS

The previous chapter discussed a mixed-initiative traded control approach to shared autonomy, demonstrating reduced completion times for a peg-in-hole problem. However, computer vision was not a robust way to achieve the target location measurement for the limited-access problem due to lighting issues. Furthermore, the confidence-based trading was not robust to false positive errors, e.g. when the localization estimate is overconfident. If not addressed, this has potential to result in damage to the workpiece or the robotic system.

To address these issues, the main contribution of this chapter is a novel method for detecting and resolving off-nominal situations using interaction forces. This chapter is published in the proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS 2018) [**?**]. This chapter also introduces shared teleoperation, which is a shift from the binary traded control-type collaborations. In shared teleoperation, both human and autonomy share control of the robot at the same time, through a haptic interface. This is effective when automation is not possible, but the autonomy can provide a *feedforward* assistance in order to offload work from the human. In this case, the human acts as a *feedback* correction in order to account for small deficiencies in the assistance to complete the task. In order to generate a haptic assistance, imitation learning from several demonstrations is used. The key to this imitation learning approach is that forces are predicted from position, meaning that assistance is provided as a function of state rather than time, which is an important consideration when sharing control with a human. Due to demonstrations that all converge to a single point, this assistance is also *compliant*, effectively acting as a nonlinear spring to guide the user to a target location.

The chapter begins by discussing the shared teleoperation system and formalizing the

off-nominal problem. Next, a method for detecting off-nominal conditions using persistent deviations of human interaction forces from expectation, i.e. from the learned compliant haptic assistance, is presented. The chapter concludes with a user study ($n = 11$) where off-nominal situations are artificially introduced during teleoperation assistance. The method reduces completion time by 17% and operator forces by 68% as compared with shared teleoperation that does not address off-nominal situations.

4.1 Problem Formulation

This section provides background on the collaborative manufacturing system and discusses how off-nominal events cause issues in this framework.

4.1.1 System Description

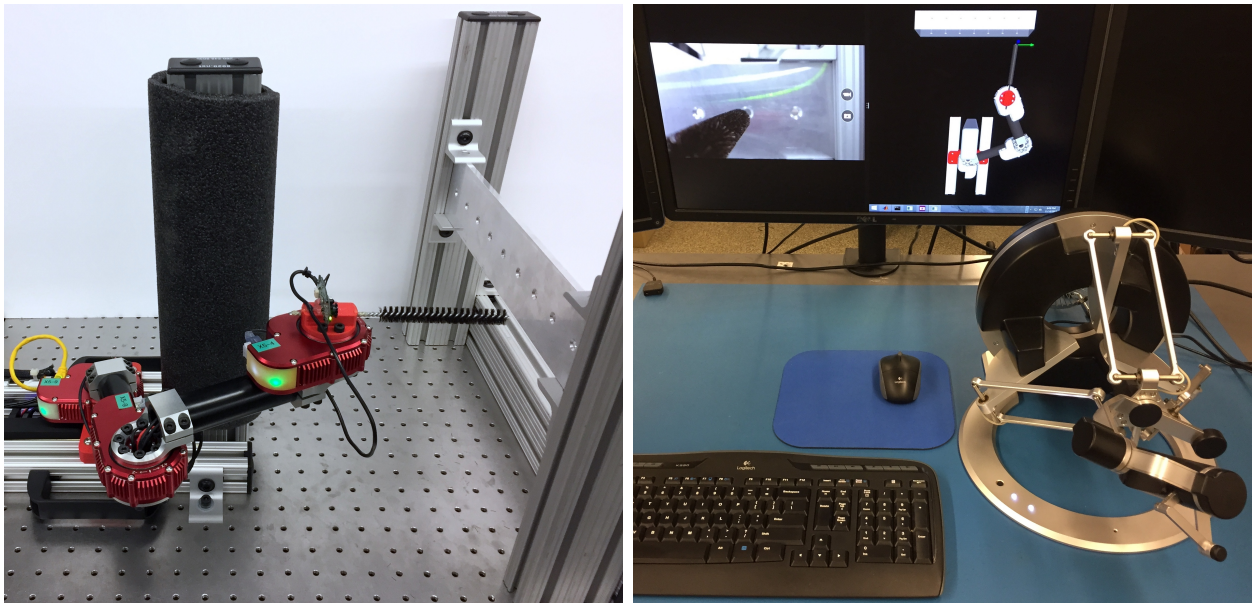
Consider the bilateral teleoperation [**?**] system shown in Fig. 4.1, consisting of 6 DoF (degree of freedom) force/torque control for both a haptic interface and a robot manipulator. Let $\boldsymbol{x}_r, \dot{\boldsymbol{x}}_r, \ddot{\boldsymbol{x}}_r \in \mathbb{R}^6$ denote the robot tool pose (position/orientation) and associated derivatives, and $\boldsymbol{x}_h, \dot{\boldsymbol{x}}_h, \ddot{\boldsymbol{x}}_h \in \mathbb{R}^6$ denote the haptic master device pose and associated derivatives. Subscripts $\boldsymbol{f}_r, \boldsymbol{f}_h \in \mathbb{R}^6$ denote forces/torques applied to the robot tool and haptic device, respectively.

Manipulator dynamics

The manipulator dynamics in task space are

$$\boldsymbol{M}(\boldsymbol{x}_r)\ddot{\boldsymbol{x}}_r + \boldsymbol{C}(\dot{\boldsymbol{x}}_r, \boldsymbol{x}_r)\dot{\boldsymbol{x}}_r + \boldsymbol{V}(\boldsymbol{x}_r) = \boldsymbol{f}_r^* + \boldsymbol{f}_r^e \tag{4.1}$$

where $\boldsymbol{f}_r^*$ is the control force/torque applied to the robot tool and $\boldsymbol{f}_r^e$ are the forces from the environment that act on the robot. $\boldsymbol{M}(\boldsymbol{x}_r), \boldsymbol{C}(\dot{\boldsymbol{x}}_r, \boldsymbol{x}_r) \in \mathbb{R}^{6\times 6}$ are mass and centrifugal/Coriolis matrices, respectively and $\boldsymbol{V}(\boldsymbol{x}_r) \in \mathbb{R}^6$ is the force due to gravity. Desired actuator torques are computed from applied force $\boldsymbol{f}_r^*$ using the operational space approach [**?**].



(a) Robotic manipulator (b) Teleoperation workstation

Figure 4.1: The collaborative manufacturing system consists of (a) a force-controlled robotic manipulator for performing a remote bottle brush operation near potentially unknown obstacles, and (b) a teleoperator workstation with a haptic interface for teleoperating the robot, haptic and visual simulation, and live camera feedback local to the robot tool.

The applied control $\boldsymbol{f}_r^*$ in (4.1) is computed from the designed impedance relationship

$$\boldsymbol{f}_r^* = \boldsymbol{K}_r(\boldsymbol{x}_h - \boldsymbol{x}_r) + \boldsymbol{B}_r(\dot{\boldsymbol{x}}_h - \dot{\boldsymbol{x}}_r) + \boldsymbol{M}_r\ddot{\boldsymbol{x}}_r, \tag{4.2}$$

where $\boldsymbol{x}_h, \dot{\boldsymbol{x}}_h$ are pose and velocity of the haptic device, $\boldsymbol{x}_r, \dot{\boldsymbol{x}}_r, \ddot{\boldsymbol{x}}_r$ are robot tool pose states, and $\boldsymbol{K}_r, \boldsymbol{B}_r, \boldsymbol{M}_r \geq 0$ are diagonal stiffness, damping and mass matrices. High stiffness, i.e. $\boldsymbol{K}_r >> 0$, increases position accuracy. High mass, i.e. $\boldsymbol{M}_r >> 0$, increases the inertia of the closed loop system. Damping $\boldsymbol{B}_r$ can be selected for critically-damped behavior as in [**?**] with respect to the desired impedance characteristics.

Haptic-device dynamics

Neglecting friction and gravity for a well-compensated haptic device, the simplified dynamics are

$$\boldsymbol{M}_h \ddot{\boldsymbol{x}}_h = \boldsymbol{f}_h^* + \boldsymbol{f}_h^o \tag{4.3}$$

where $\boldsymbol{f}_h^*$ is the control applied to the haptic device, $\boldsymbol{f}_h^o$ is the force applied by the operator, and $\boldsymbol{M}_r$ is the mass of the device. The remaining discussion considers the static case, i.e. $\boldsymbol{M}_h = 0$, because the dynamics of the haptic device are negligible compared to the forces acting on the device.

Using the position-force architecture in bilateral teleoperation [**?**], the applied haptic control $\boldsymbol{f}_h^*$ in (4.3) is

$$\boldsymbol{f}_h^* = -\lambda \boldsymbol{f}_r^*, \tag{4.4}$$

where the scalar $\lambda \geq 0$ determines to what degree the forces applied to the robot in (4.2) are reflected to the teleoperator. Typically, $\lambda << 1$ is required to maintain stability of this system, resulting in attenuated forces reflected to the operator. Stability can be analyzed using passivity [**?**].

4.1.2 Compliant Imitation Learning for Task Assistance

To assist the operator with a manufacturing task, the robot provides assistance in the form of a feedforward force applied to the haptic device. Imitation learning [**?**] is used to develop the assistance policy a priori. For each task with a known target location, it is assumed that a skilled operator demonstrates a family of trajectories in haptic simulation. Note that simulated dynamics and environmental interactions in (4.1) and force reflection in (4.4) are present in this step. The demonstrations $\mathcal{D} = \{\bar{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r, \boldsymbol{f}_h^{o,\pi}\}$ are assumed sampled from

$$\boldsymbol{f}_h^{o,\pi} \sim \mathcal{N}(\boldsymbol{\mu}(\bar{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r), \boldsymbol{\Sigma}(\bar{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r)) \tag{4.5}$$

where $\boldsymbol{\mu}(\bar{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r)$ and $\boldsymbol{\Sigma}(\bar{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r)$ are the mean and variance of the expert operator demonstrated force $\boldsymbol{f}_h^{o,\pi}$, and $\bar{\boldsymbol{x}}_r$ is the robot tool pose normalized to the known target $\boldsymbol{x}_t$ (i.e in

simulation)

$$\bar{\boldsymbol{x}}_r = \boldsymbol{x}_r - \boldsymbol{x}_t. \tag{4.6}$$

Eq. (4.5) is learned with Gaussian Mixture Regression (GMR) [**?**], using a k-means-initialized EM algorithm [**?**].

Remark 1 *Note that* (4.5) *directly encodes a compliant operator policy* $\boldsymbol{f}_h^{o,\pi}$ *in the form of a nonlinear impedance (stiffness and damping) around the deviation of robot states* $\boldsymbol{x}_r, \dot{\boldsymbol{x}}_r$ *from the target position* $\boldsymbol{x}_t$ *rather than a pure position trajectory, and that the demonstration is assumed to be time-independent as in [**?**].*

Online haptic assistance is provided by modifying (4.4)

$$\boldsymbol{f}_h^* = -\lambda \boldsymbol{f}_r^* + \boldsymbol{f}_h^a \tag{4.7}$$

to include the new assistive force input $\boldsymbol{f}_h^a$, computed from the expectation of the demonstration in (4.5)

$$\boldsymbol{f}_h^a = \gamma \mathbb{E}\left[\boldsymbol{f}_h^{o,\pi} | \hat{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r\right] \tag{4.8}$$

where $1 \geq \gamma \geq 0$ is a gain that influences the degree of assistance offered to the operator, i.e. the *level of assistance*, and $\hat{\boldsymbol{x}}_r$ is the robot tool pose normalized as in (4.6), but now to an estimate of the target position $\hat{\boldsymbol{x}}_t$.

Remark 2 *When* $\gamma = 1$*, assistance is always provided. Previous works [**?**, **?**, **?**] have proposed scaling the level of assistance* γ *based on confidence (or variance) in the haptic assistance*

$$\gamma \propto conf\left(\boldsymbol{f}_h^{o,\pi} | \hat{\boldsymbol{x}}_r, \dot{\boldsymbol{x}}_r\right). \tag{4.9}$$

In this manner, confidence in assistance results in more assistance. The objective is to provide a similar method for scaling the level of assistance γ*, but for managing off-nominal conditions that differ from those used to develop the haptic assistance.*

4.1.3 Problem Statement

Because the assistance policy (4.8) is developed under nominal conditions (i.e. known target, task, and environment), it imitates the demonstrator under consistent conditions. A challenge arises in off-nominal situations (i.e. uncertain target, task, or environment) since the original demonstration no longer accurately predicts the most likely operator action. In this event, the operator may fight the assistance, especially in regions where the learned policy exhibits low compliance, e.g. in regions of low variance [**?**, **?**] or regions of high confidence [**?**]. The central problem examined by this chapter is thus how to attenuate the haptic assistance in off-nominal situations in order to improve collaboration performance.

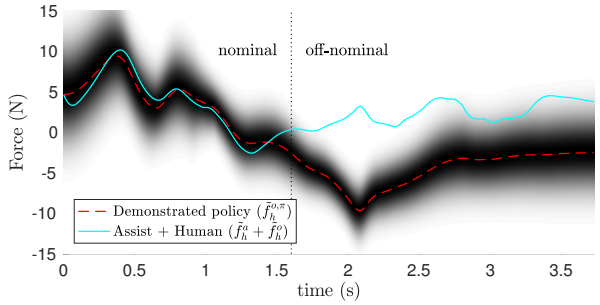
4.2 Methodology

This section introduces the adaptive assistance approach to handle off-nominal situations in the learned task compliance paradigm.

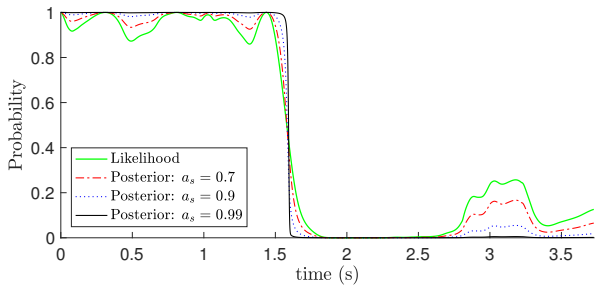
4.2.1 Solution Approach

The learned behavior (from demonstration) in (4.5) is leveraged to classify human actions as nominal or off-nominal. When off-nominal, assistance is attenuated to provide the operator sufficient authority to address the anomaly. This concept is depicted in Fig. 4.2a. Prior to t = 1.6 s, the combination of human actions $\boldsymbol{f}_h^o$ and assistance $\boldsymbol{f}_h^a$ closely matches the learned policy $\boldsymbol{f}_h^{o,\pi}$ (density shaded for reference). At time t = 1.6 s, the human begins to exert actions that diverge from the expectation. Fig. 4.2c shows the attenuation of the level of assistance γ under off-nominal behavior.

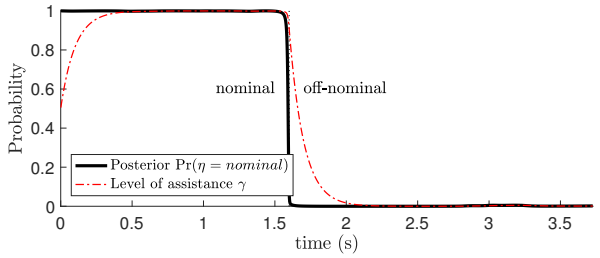
The following subsections address the resulting challenges: *detection* - how to perform classification of operator actions as nominal or off-nominal; and *attenuation* - how to use the classification result to attenuate the assistance.



(a) Evolution of assistance and operator actions over time



(b) Likelihood and posterior for choices of $\boldsymbol{A}$



(c) Posterior and level of assistance

Figure 4.2: The proposed modified learned compliance (M-LC) classifies filtered operator actions $\tilde{\boldsymbol{f}}_h^o$ as either nominal or off-nominal. (a) At time $t = 1.6$ s, the operator begins applying actions that diverge from the learned compliant behavior (density in grayscale, centered around the demonstration $\tilde{\boldsymbol{f}}_h^{o,\pi}$). (b) The likelihood $\mathbf{Pr}(\tilde{\boldsymbol{f}}_h^o)$ quantifies this divergence, making it a useful indicator for classifying behavior as nominal or off-nominal. (c) The level of assistance γ is a smoothed version of the posterior $\Pr(\eta = nominal)$.

4.2.2 Pre-filtering of data

Because human actions are used to detect off-nominal behavior, data is filtered to remove signal elements that are higher frequency than human reaction times. For the vector of signals, $\boldsymbol{\xi} = (\boldsymbol{f}_h^{o\top}, \boldsymbol{f}_h^{a\top}, \boldsymbol{x}_r{}^\top, \dot{\boldsymbol{x}}_r^\top)^\top$, this is achieved with a first-order lowpass filter

$$\tau\dot{\tilde{\boldsymbol{\xi}}} + \tilde{\boldsymbol{\xi}} = \boldsymbol{\xi}, \tag{4.10}$$

with bandwidth parameter τ, and filtered signals $\tilde{\boldsymbol{\xi}}$. The filter's time constant is selected as $\tau = 0.1$ s to match the typical human response time of 100 ms [**?**].

4.2.3 Classifying Off-Nominal Human Behavior

To classify filtered operator actions $\tilde{\boldsymbol{f}}_h^o$ as nominal or off-nominal, consider a latent discrete random variable $\eta = \{nominal, \neg nominal\}$ that influences human actions. The driver η is assumed a Markov parameter, and its relationship between applied assistance $\tilde{\boldsymbol{f}}_h^a$ and operator forces $\tilde{\boldsymbol{f}}_h^o$ is graphically modeled in Fig. 4.3. Note the vector $\mathbf{Pr}(\eta)$ contains the probability of both $\eta = nominal$ and $\eta = \neg nominal$, and that $\Pr(\eta = \cdot)$ indicates the probability of being in one of these states. Under these assumptions, the latent driver η is inferred using standard recursive Bayesian state estimation [**?**] for time interval k

$$\mathbf{Pr}(\eta_k|\eta_{k-1}) \propto \mathbf{Pr}(\tilde{\boldsymbol{f}}_{h,k}^o|\tilde{\boldsymbol{f}}_{h,k}^a, \eta_k) \sum_\eta \boldsymbol{\mathcal{T}}(\eta_k|\eta_{k-1})\, \mathbf{Pr}(\eta_{k-1}), \tag{4.11}$$

where $\mathbf{Pr}(\eta_{k-1})$ is the prior of η, $\boldsymbol{\mathcal{T}}(\eta_k|\eta_{k-1})$ is the state transition model, $\mathbf{Pr}(\tilde{\boldsymbol{f}}_{h,k}^o|\tilde{\boldsymbol{f}}_{h,k}^a, \eta_k)$ is the likelihood of observing filtered human actions $\tilde{\boldsymbol{f}}_h^o$ given filtered applied assistance $\tilde{\boldsymbol{f}}_h^a$ and the driver η, and $\propto$ indicates normalization such that $\sum_\eta \mathbf{Pr}(\eta_k) = 1$. Fig. 4.2b shows several posteriors and the likelihood for the actions in Fig. 4.2a. Note that at time $t = 1.6$ s, the posterior estimate of nominal behavior becomes nearly 0, as desired.

Likelihood model

To compute the likelihood, it is assumed that nominal behavior ($\eta = nominal$) corresponds to human actions expected by the model learned from demonstration, and that off-nominal

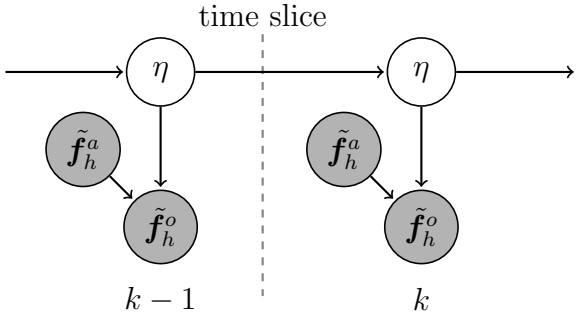


Figure 4.3: The relationship between latent driver η, applied assistance $\tilde{\boldsymbol{f}}_h^a$ and operator forces $\tilde{\boldsymbol{f}}_h^o$ is modeled as a Dynamic Bayesian Network (DBN), with two time steps shown for time interval k. Observable states are shaded.

($\eta = \neg nominal$) behavior corresponds to any human actions outside of this expectation. Under this assumption, the likelihood of observing the filtered human action $\tilde{\boldsymbol{f}}_h^o$

$$\mathbf{Pr}(\tilde{\boldsymbol{f}}_h^o | \tilde{\boldsymbol{f}}_h^a, \eta) = \begin{cases} K(\tilde{\boldsymbol{f}}_h^o) & \text{if } \eta = nominal \\ 1 - K(\tilde{\boldsymbol{f}}_h^o) & \text{if } \eta = \neg nominal \end{cases} \tag{4.12}$$

where $K(\tilde{\boldsymbol{f}}_h^o)$ is a normalized multivariate Gaussian

$$K(\tilde{\boldsymbol{f}}_h^o; \breve{\boldsymbol{\mu}}, \breve{\boldsymbol{\Sigma}}) = \exp\left\{ -\frac{1}{2} (\tilde{\boldsymbol{f}}_h^o - \breve{\boldsymbol{\mu}})^\top \breve{\boldsymbol{\Sigma}}^{-1} (\tilde{\boldsymbol{f}}_h^o - \breve{\boldsymbol{\mu}}) \right\} \tag{4.13}$$

where $\breve{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}(\tilde{\bar{\boldsymbol{x}}}_r, \tilde{\dot{\boldsymbol{x}}}_r)$ is the variance of demonstrated operator forces in (4.5), and $\breve{\boldsymbol{\mu}} = \boldsymbol{\mu}(\tilde{\bar{\boldsymbol{x}}}_r, \tilde{\dot{\boldsymbol{x}}}_r) - \tilde{\boldsymbol{f}}_h^a$, is the difference between mean demonstrated operator forces $\boldsymbol{\mu}(\tilde{\bar{\boldsymbol{x}}}_r, \tilde{\dot{\boldsymbol{x}}}_r)$ in (4.5) and applied assistance $\tilde{\boldsymbol{f}}_h^a$.

Remark 3 *The kernel* (4.13) *encodes how the operator is expected to behave. The assumption is that the operator is expected to match the demonstrated policy in* (4.5)*. In this manner, the operator is expected to provide a force* $\tilde{\boldsymbol{f}}_h^o$ *in addition to the force provided by assistance* $\tilde{\boldsymbol{f}}_h^a$ *to match the mean of the demonstration* $\boldsymbol{\mu}(\tilde{\bar{\boldsymbol{x}}}_r, \tilde{\dot{\boldsymbol{x}}}_r)$*, i.e.*

$$\mathbb{E}\left[\tilde{\boldsymbol{f}}_h^o + \tilde{\boldsymbol{f}}_h^a - \boldsymbol{\mu}(\tilde{\bar{\boldsymbol{x}}}_r, \tilde{\dot{\boldsymbol{x}}}_r) \right] = \mathbf{0}. \tag{4.14}$$

If this is not true, i.e. within the variance of the demonstration $\boldsymbol{\Sigma}(\tilde{\tilde{\boldsymbol{x}}}_r, \tilde{\dot{\boldsymbol{x}}}_r)$*, the operator is exhibiting off-nominal behavior. This is the principal mechanism used to classify operator behavior as nominal or off-nominal. Fig. 4.2b shows the likelihood* $\Pr(\tilde{\boldsymbol{f}}_h^o | \eta = nominal)$ *for the actions in Fig. 4.2a.*

State transition model

To predict the evolution of the driver η, consider a linear first-order Markov transition model

$$\boldsymbol{\mathcal{T}}(\eta_k | \eta_{k-1}) = \boldsymbol{A}\, \mathbf{Pr}(\eta_{k-1}), \tag{4.15}$$

where the state transition matrix $\boldsymbol{A}$ predicts the current probability vector of the driver η_k from the previous η_{k-1}. Note that $\boldsymbol{A}$ is expected to be heavily weighted on the diagonal, because at each time step k, the probability of staying in the same state is expected to be far higher than the probability of transitioning to the alternate state, e.g.: from $\eta = nominal$ to $\eta = \neg nominal$.

While it is possible to learn $\boldsymbol{A}$ from data using the Baum-Welch algorithm [**?**], this presents a challenge to the designer. If the data is absent of transitions between off-nominal and nominal, the algorithm is likely to learn a completely diagonal matrix and thus never predict transitions between states. Conversely, an experiment designed to mimic a pre-specified set of "off-nominal" situations will produce a transition model specific to that experiment, and may not generalize well to other situations.

Selection of the transition model

An alternate approach is to design the transition model based on knowledge of human behavior. Consider a symmetric transition matrix

$$\boldsymbol{A} = \begin{pmatrix} a_s & a_t \\ a_t & a_s \end{pmatrix} \tag{4.16}$$

where a_s is the probability of staying in the current state, and a_t is the probability of transitioning to the alternative state, e.g. from $\eta = nominal$ to $\eta = \neg nominal$. Fig. 4.2b shows the effect of choice of a_s on the posterior $\Pr(\eta = nominal)$. This work assumes that human-directed transitions to a new state will occur at a frequency that is equal to or lower than the human response time, i.e. $\tilde{\tau} a_t \leq \Delta$ where Δ is the discrete sample rate of the system, and $\tilde{\tau}$ is the human response time of 100 ms [**?**]. Selecting the equivalence for a conservative model, i.e. $\tilde{\tau} a_t = \Delta$, yields the matrix design criterion $a_t = \Delta/\tilde{\tau}$ and $a_s = 1 - a_t$.

4.2.4 Smooth Transitions

In order to reduce abrupt changes in haptic assistance felt by the operator, the level of assistance γ applied in (4.8) is a smoothed version of the posterior $\Pr(\eta = nominal)$,

$$\tau \dot{\gamma} + \gamma = \Pr(\eta = nominal). \tag{4.17}$$

A time constant of $\tau_s = 0.1$ s is selected, again because the typical human response time is known to be 100 ms [**?**]. Fig. 4.2c shows the level of assistance γ. Note that although stability is not guaranteed during transitions, this smoothing ensures that the response remains within human control bandwidth.

4.3 Experiment and Results

4.3.1 Bottle brush Cleaning Operation

This experiment studies a remote hole cleaning operation using a bottle brush mounted to a robot arm. The 3 DoF planar robot arm, driven by HEBI actuators, allows for tool motion in two states $x_1,\ x_2$ with orientation controlled at a fixed constant. The task consists of inserting the bottle brush tip (via teleoperation) into an intended target pilot hole. The human teleoperates the arm from a remote workstation using a Force Dimension Omega.7 haptic device, and is provided a 3D visual and haptic environment (using CHAI3D [**?**]) with real-time robot pose information (using CoreRobotics [**?**]). The target pilot hole is indicated

by a red box in the visualization. A webcam mounted to the last robot link additionally provides a close up view of the bottle brush tool and local environment. The control rate is $\Delta = 1$ ms.

4.3.2 Models and Parameter Selection

An expert operator demonstrates twenty-eight trajectories in nominal simulation to train the imitation learning policy in (4.5). The trajectory initial conditions are selected to span the robot workspace and provide information regarding motion away from other holes, movement along the panel containing holes, and extremities of the robot reach. Training is performed with 5% of the data and cross-validation with 20% of the data. The remaining 75% of the data are used for testing. $K = 8$ clusters are used for the Gaussian mixture centers based on Pareto analysis of the regression mean squared error (mse). Training data, regression mean, variance, and mixture centers are shown in Fig. 4.4.

The bilateral stiffness gain $\boldsymbol{K}_r$ is chosen to be 400 N/m in translation and 10 N/rad in orientation. Damping gain $\boldsymbol{B}_r$ is chosen to be 3 N-s/m in translation and 0.1 N-s/rad in orientation. Because the workspace of the haptic device is amplified for the task, the force reflection gain λ in (4.4) is chosen to be 0.5 for translation and 0 for orientation since the haptic device is only active in translation. This choice results in half of the force applied to the robot tool being reflected back to the teleoperator. The transition matrix in (4.16) is defined with $a_t = 0.01$ and $a_s = 0.99$ based on the sample rate $\Delta = 1$ ms, and rationale in Section 4.2.3.

4.3.3 Human Study Procedures

Eleven ($n = 11$) subjects were asked to perform the teleoperated bottle brush insertion operation for three scenarios, shown in Fig. 4.5: one nominal, and two off-nominal. The two off-nominal scenarios are artificially introduced into the control algorithm and visible to the operator in the runtime virtualization. Before taking data, the test is described, and the subject performs the nominal task with and without assistance several times to achieve a

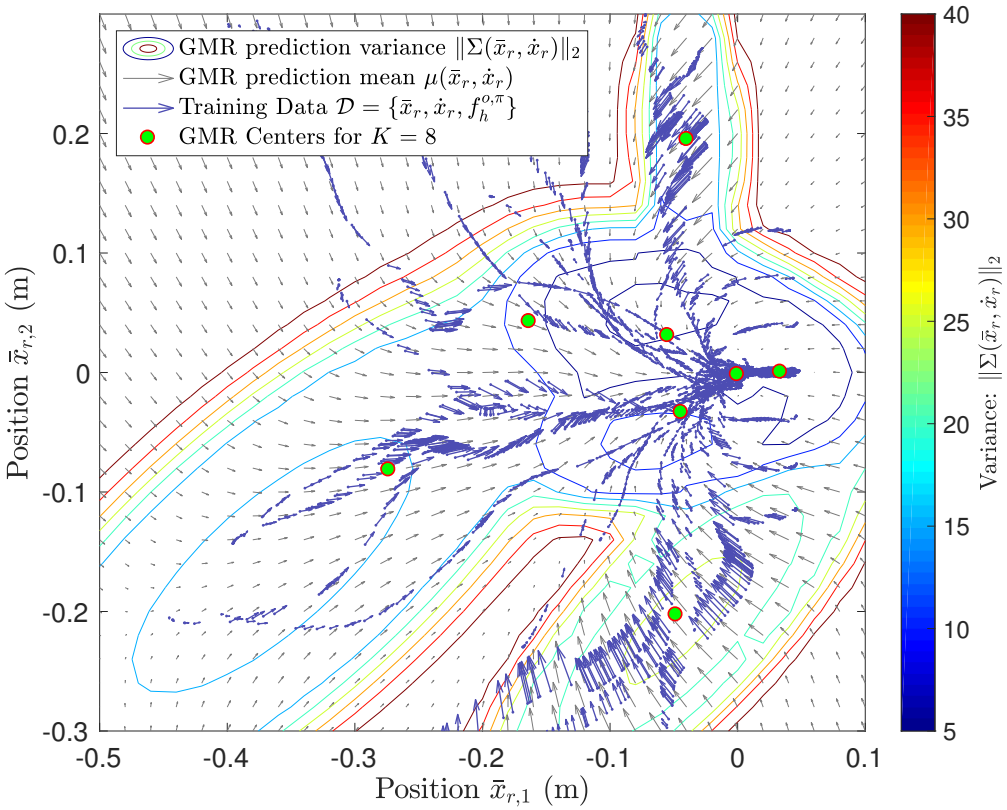
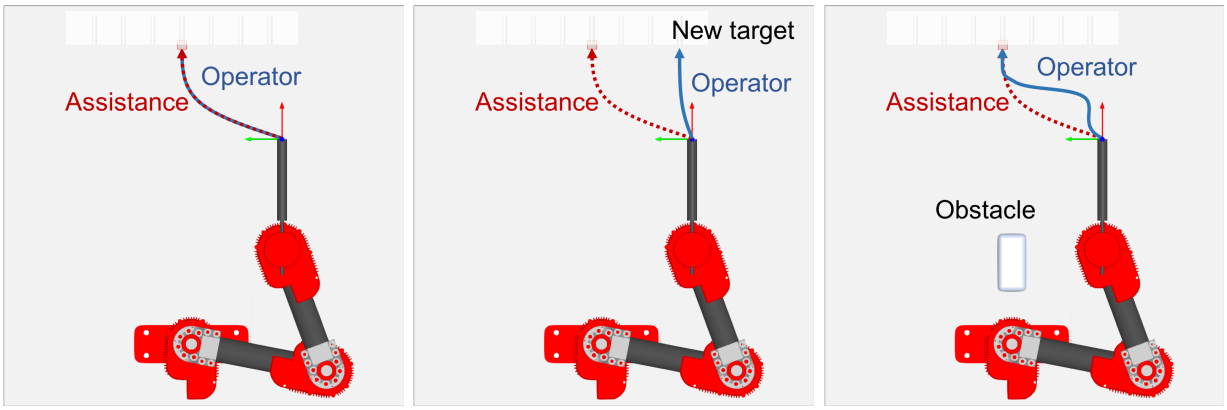


Figure 4.4: The learned policy $\boldsymbol{f}_h^{o,\pi}$ from (4.8) is shown for the bottle brush operation for the 2 primary dimensions of motion. Training data $\mathcal{D}$ is overlaid for reference. Note that the data is normalized such that the hole location is at $\bar{\boldsymbol{x}}_r = \mathbf{0}$. $K = 8$ clusters were used to fit the data.

baseline level of familiarity. This is to ensure the results compare the effects of the proposed method rather than subject familiarity with basic operation, since users are expected to be familiar with the system in practice. The subject performs each of the nine total test cases fives times. For each trial, the subject is instructed to perform the test as quickly as is comfortably possible. When the robot is offering assistance, the subject is instructed to let the robot help as much as possible.

Nominal. In nominal operation in Fig. 4.5a, the operator completes the bottle brush



(a) Nominal (b) Switched target (c) Obstacle

Figure 4.5: The proposed method is evaluated for three scenarios. (a) The nominal case has the same conditions as training, although the real robot is used instead of simulated dynamics. (b) A switched target requires the operator to oppose the assistance at the end. (c) An obstacle requires the operator to oppose the assistance to navigate around the obstacle.

insertion task under the same conditions as the policy training phase (with the exception being that training occurred with simulated dynamics and experiments use the real robot). Each subject performs the task five times each for three modes of operation: (i) *no assistance* (NA), (ii) assistance using the *learned compliance* (LC), and (iii) assistance using the proposed approach, i.e. *modified learned compliance* (M-LC).

Switched target. In this scenario in Fig. 4.5b, the target location is switched from the nominal target location. This requires the operator to oppose the assistance in order to pull the robot away from the nominal target toward the new target. This off-nominal condition exists for the duration of each test. As in the nominal case, each subject performs the task five times for each of the three modes of operation.

Obstacle. In this scenario in Fig. 4.5c, an obstacle is introduced that requires the operator to temporarily oppose the assistance to pull the robot away from the obstacle. Once the robot is clear from the obstacle, the operator should align with the assistance.

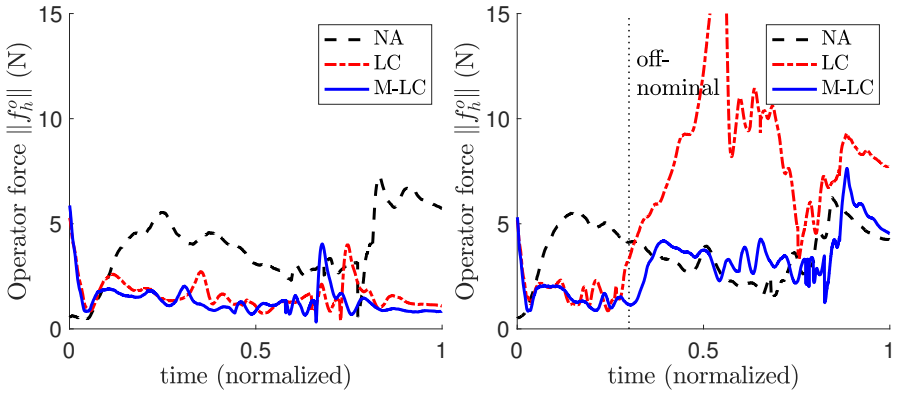
Table 4.1: Performance comparison for $n = 11$ subjects: no assistance (NA), learned compliance (LC), and the proposed modified learned compliance (M-LC) for one nominal scenario and two off-nominal scenarios. The first three rows indicate mean μ and standard deviation σ for all test subjects, and colors indicate which method performed best (green) and worst (red) in the corresponding metric. The bottom two rows show mean percentage individual subject improvement using M-LC over NA and LC. The margin of error ME is for a 90% confidence interval, with significant results highlighted in blue.

	Nominal		Switched target		Obstacle	
Method	T **(s)**	$F_{o,rms}$ **(N)**	T **(s)**	$F_{o,rms}$ **(N)**	T **(s)**	$F_{o,rms}$ **(N)**
NA ($\mu \pm \sigma$)	4.1±1.5	3.8±0.9	3.9±1.6	3.6±1.2	4.6±1.3	3.4±1.4
LC ($\mu \pm \sigma$)	2.7±0.8	3.1±1.6	4.4±1.7	13.6±2.0	3.8±0.9	4.3±1.4
M-LC ($\mu \pm \sigma$)	2.8±0.7	3.1±1.4	3.4±1.2	4.5±3.6	3.5±0.9	3.5±1.3
M-LC over NA ($\mu \pm ME$)	27±12%	17±12%	8±12%	-19±20%	22±5%	-7±13%
M-LC over LC ($\mu \pm ME$)	-3±9%	-6±10%	17±10%	68±10%	7±6%	17±7%

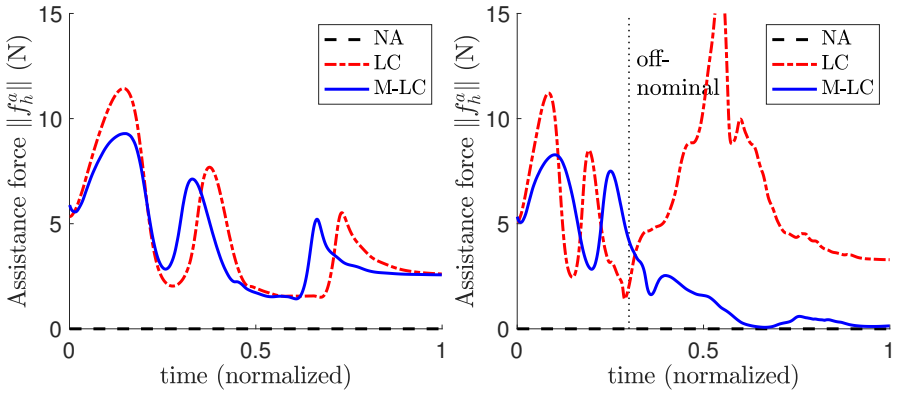
This off-nominal condition exists for roughly 0.5s while the robot is near the obstacle. As in the previous cases, each subject performs the task five times for each of the three modes of operation.

4.3.4 Performance Evaluation

Results are summarized in Tbl. 4.1. Operation completion time and root mean square (RMS) of forces from the operator are used to evaluate the system performance. Completion time is important to consider for manufacturing throughput in the motivating application. The



(a) Nominal case, operator force (b) Off-nominal case, operator force



(c) Nominal case, assistance force (d) Off-nominal case, assistance force

Figure 4.6: A comparison of operator input magnitude $\|\boldsymbol{f}_h^o\|_2$ and assistance input magnitude $\|\boldsymbol{f}_h^a\|_2$ with no assistance (NA), learned compliance (LC), and the proposed modified learned compliance (M-LC) for (a,c) nominal operation, and (b,d) an off-nominal case (switched target hole). In nominal situations, both modes of assistance (LC and M-LC) reduce the magnitudes of forces exerted by the operator for task completion. However, in off-nominal situations, learned compliance (LC) significantly increases the forces required by the operator because the operator opposes the assistance actions. The proposed method (M-LC) detects the off-nominal behavior and attenuates the assistance to allow the operator to complete the task with no assistance, achieving the performance of the baseline in these conditions.

root mean square (RMS) of operator forces for a single task of time T, defined as

$$F_{o,rms} = \sqrt{\frac{1}{T}\int_0^T \boldsymbol{f}_h^o(t)^\top \boldsymbol{f}_h^o(t)\ dt}, \tag{4.18}$$

is in importance indicator for the amount of effort required by the operator for task completion. Successful assistance reduces both the mean completion time T and RMS of the operator forces $F_{o,rms}$.

4.4 Discussion

Examining the results in Table 4.1, no assistance (NA) performs the poorest in the nominal situation ($T = 4.1s$, $F_{o,rms} = 3.8N$), as expected. Learned compliance (LC) and the proposed modified learned compliance (M-LC) perform similarly for this case ($T = 2.7s$, $F_{o,rms} = 3.1N$). In the off-nominal scenarios, the operator force is highest for LC due to opposition from assistance. The proposed M-LC exhibits the lowest completion times in the off-nominal scenarios ($T = 3.4s,\ T = 3.5s$) because the method enables assistance in nominal regimes but quickly allows the operator to deviate without prolonged opposition from the assistive forces.

4.4.1 Reduction in forces

The proposed M-LC reduces interaction forces over LC for both the obstacle ($F_{o,rms} = 17\%$) and switched target cases ($F_{o,rms} = 68\%$). The improvement is most apparent in the switched target due to high operator forces needed to counteract the opposing policy. Figure 4.6 examines this off-nominal scenario (compared to the nominal case) that causes the assistance to severely degrade performance. At time $t = 0.3$, the off-nominal event becomes apparent, i.e. the operator intends to move toward the switched target. Because the assistance is still directed toward the original goal, the learned compliance (LC) forces $\|\boldsymbol{f}_h^a\|$ peak in opposition to the operator forces $\|\boldsymbol{f}_h^o\|$. To handle this scenario, the proposed modified learned compliance (M-LC) detects and classifies the human behavior as off-nominal and attenuates the assistance accordingly. As a result, the operator forces $\|\boldsymbol{f}_h^o\|$ are consistent

with the original operator forces in the no assistance (NA) case since assistance is no longer provided.

4.4.2 Reduction in completion time

M-LC improves completion time over LC for the switched target ($T = 17\%$), due to the additional time subjects take to counteract the opposing policy. M-LC also improves over NA for the obstacle ($T = 22\%$) and the nominal case ($T = 27\%$). Improvements are not statistically significant in the other categories.

4.4.3 Subject preference

Subject preference appears to slightly degrade performance of M-LC over LC in nominal operation ($T = -3 \pm 9\%$, $F_{o,rms} = -6 \pm 10\%$), although the results are not significant. Figure 4.7 shows the level of assistance γ for the nominal case. Off-nominal conditions were detected for three of the subjects in the nominal case; one of these subjects indicated the system "gave up" if they pushed or pulled on the assistance. In these cases, M-LC detected an off-nominal condition due to disagreement between the assistance and operator, for which M-LC gives priority to the operator.

4.4.4 Bidirectional transitions

The obstacle scenario is interesting because the operator temporarily exhibits off-nominal behavior to circumnavigate the obstacle before returning to nominal. This case demonstrates the robustness of the method for returning to nominal operation. Figure 4.8 shows the distribution of the level of assistance γ for the proposed M-LC in the obstacle scenario. Note that around $t = 0.4$, assistance is attenuated due to the operator navigating away from the obstacle and returns to nominal after the obstacle is away from the robot. Subjects reported that the M-LC method would "recognize" when they wanted to complete the task and return to providing assistance after the obstacle was circumnavigated.

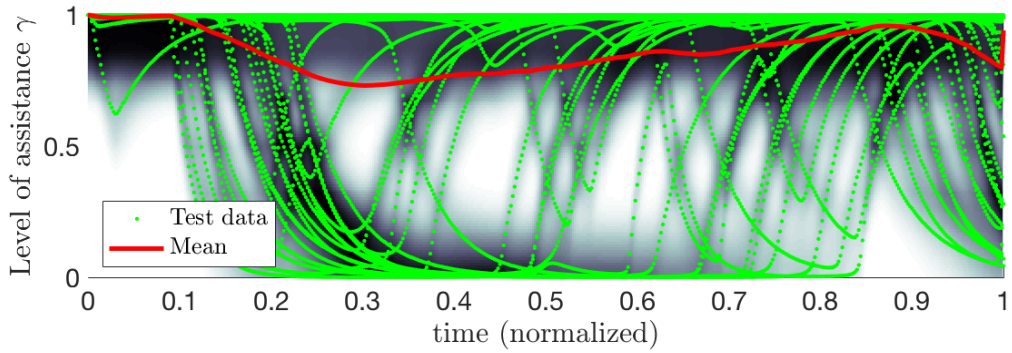


Figure 4.7: Level of assistance γ using the proposed method (M-LC) for the nominal scenario. Some subjects reported the assistance did not match their speed preference, hence several test cases exhibited attenuated assistance.

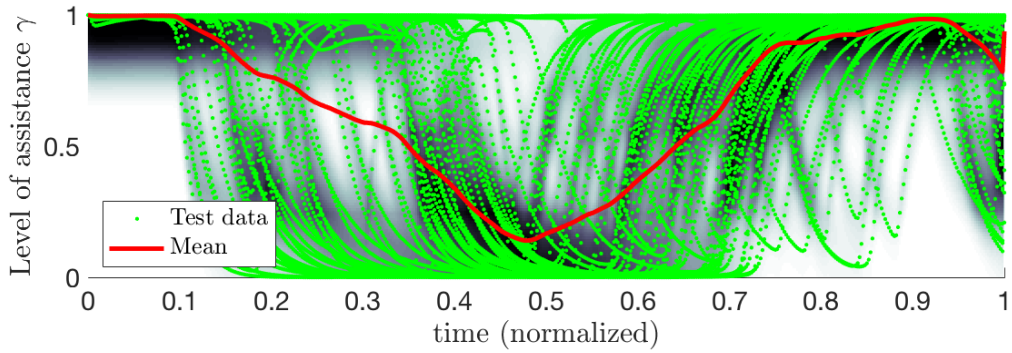


Figure 4.8: Level of assistance γ using the proposed method (M-LC) for the obstacle scenario. Note that the subjects circumnavigate the obstacle near $t = 0.4$ s. Darker regions indicate higher density of experimental data.

4.5 Conclusion

This chapter proposed a method for off-nominal identification in order to attenuate assistance in shared teleoperation. The method exploits the probabilities of expected operator forces learned from demonstration. Without this method, teleoperation assistance opposes operator motions, resulting in increased operator forces when off-nominal events arise during a task.

Experimental results show that the proposed method reduces completion times by 17% and operator forces by 68% as compared with shared teleoperation without the method. Further, the method does not result in any statistically significant changes in performance in nominal situations as compared with shared teleoperation without the method.

There are several limitations with the methods presented in this chapter.

- The assistance is trained in haptic simulation, however there is a discrepancy between simulation physics and reality. To provide a good assistance, the simulation must be sufficiently close to reality, however *how close* is not discussed. Since the trained assistance improves performance on the physical system, the discrepancy for this work is acceptable, but to learn full automation from demonstration, higher fidelity models or kinesthetic demonstrations are needed.

- The developed compliant assistance is useful as an aide to the teleoperator to help drive the system to the hole location, but it does not help out with cleaning itself. The imitation learning techniques utilized to provide assistance are insufficient to automate the cleaning procedure. Thus for full automation, more sophisticated imitation learning techniques are needed.

- The method uses interaction forces between the human and the robot to detect off-nominal conditions. This is less of a problem than in the previous Chapter 3, since shared teleoperation means the human is continuously in the loop with the haptic assistance. However, predictions from assistance alone are not sufficient to indicate off-nominal conditions for other interaction forces, i.e. physical interactions between the robot and the environmental. In principal, the same off-nominal detection technique can be applied, provided a model of expected physical interaction forces during the task exists.

Chapter 5
LEARNING POLICIES FOR TASK AUTOMATION

This chapter proposes a method for imitation learning from human demonstrations of the task, e.g. a brush cleaning operation shown in Figure 5.1. The objective is to learn a model of task dynamics (i.e. autonomous dynamical motion and interaction force predictions) that can be used to develop a control policy π that replicates the demonstrated motion and forces. The main challenge is that the brush cleaning task exhibits motions that do not have well-defined equilibrium points, and complex physical interaction forces such as friction hysteresis. These properties properties do not allow for the direct application of existing dynamical system imitation learning techniques.

The main contribution of this chapter is the development of a task dynamics model structure that addresses these data complexities, and a state-action decomposition procedure for finding model realizations with (i) the popular Gaussian mixture regression (GMR) technique for predicting motion and interaction forces (i.e. *actions*), and (ii) nonlinear autoregressive (NARX) neural networks for learning the sequence of these actions. Because the procedure produces many candidate models, a method for model selection is presented using information criterion. The chapter concludes with a trajectory optimization method for developing model-reference motion control policies.

5.1 Background

Consider the training datasets (kinesthetic and haptic) of 15 trials each presented in Figure 5.2. Note that this data is shown for the principal stroke cycle (coaxial with the hole), since this is the majority of the motion during the task. There are several interesting features that appear in both datasets and introduce challenges for learning a good model of the data.

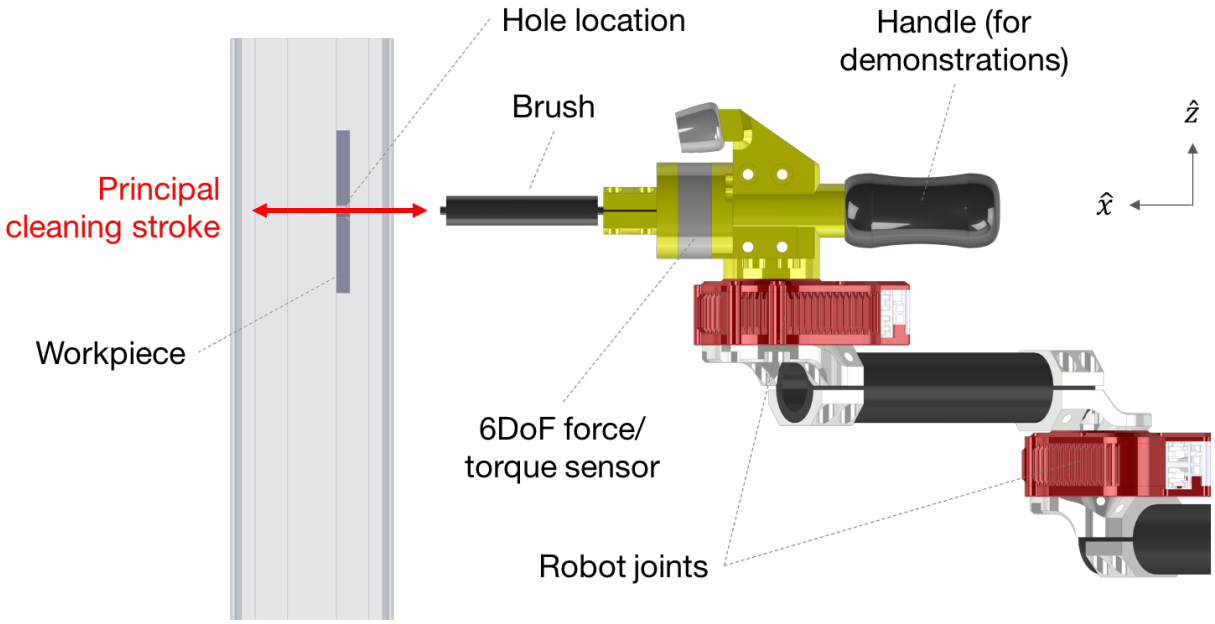
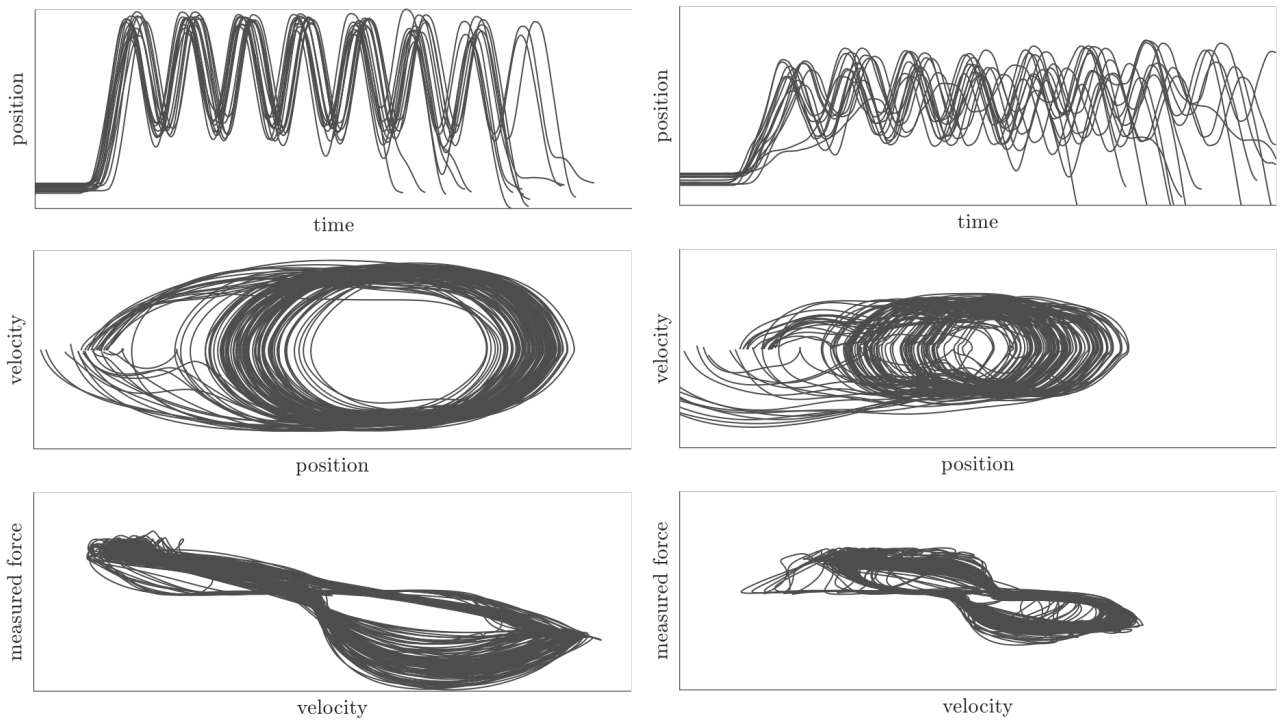


Figure 5.1: In a cleaning task, a brush mounted to a robot is used to clean a hole in a nearby workpiece with a repetition of several cleaning stroke cycles. An expert user demonstrates this task kinesthetically using the handle on the tool. A force/torque sensor measures interaction forces of the brush with the workpiece, and the robot joints provide kinematic feedback (positions, velocities, and accelerations) of the tool.

5.1.1 Temporal inconsistency

The first feature examined is the temporal inconsistency found in both datasets. Consider the position-time plots in Figure 5.2. While the kinesthetic data shows fairly consistent position behavior over time, the number of cleaning cycles varies between demonstrations. Examining the haptic dataset, the variation of position over time becomes noticeable: cycle peaks occur with increasing uncertainty over time. As with the kinesthetic dataset, the number of cycles is inconsistent between trials.

This introduces a challenge in addressing the time-dependency during learning. Some



(a) Kinesthetic demonstrations

(b) Haptic demonstrations

Figure 5.2: Data from the same expert are shown for both kinesthetic and haptic demonstrations. Each dataset consists of 15 trials. Note plot axes scales are consistent for both demonstrations.

approaches, including [?, ?, ?, ?], predict motion as a function of time[1], i.e.:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}(t), \boldsymbol{\Sigma}(t)). \tag{5.1}$$

There are two limitations with this approach. First, the data must be time-aligned to generate good estimates for mean $\boldsymbol{\mu}(t)$ and variance $\boldsymbol{\Sigma}(t)$. However, popular temporal alignment techniques such as Dynamic Time Warping (DTW) are known perform poorly with the stochastic data found in both datasets, specifically losing the global shape of trajectories [?]

[1]Note that Gaussian assumptions are a popular choice due to their computationally simplicity and linear properties, making them desirable for probabilistic learning and control problems.

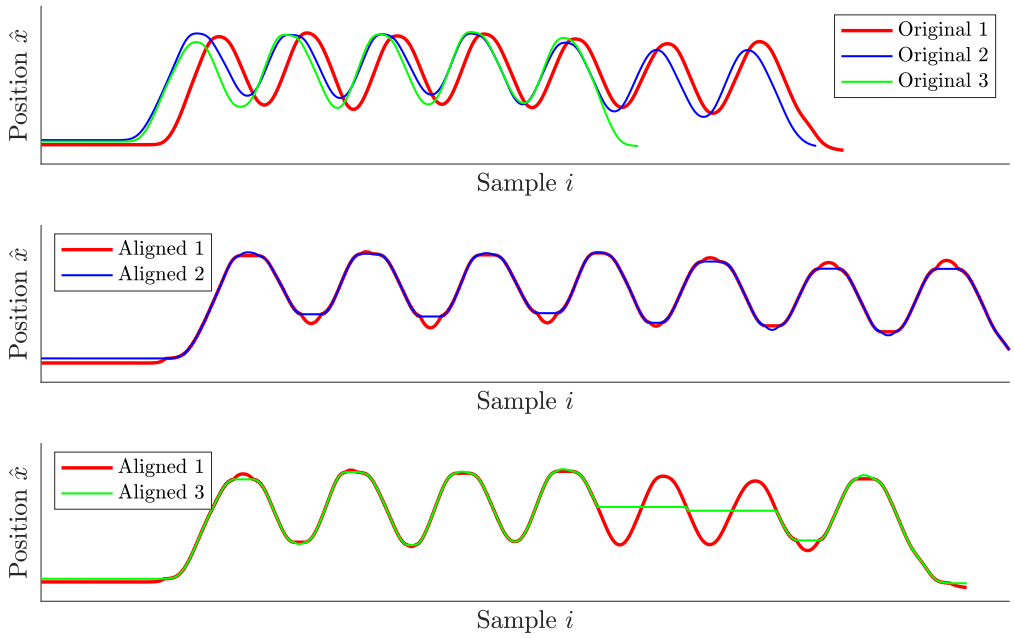


Figure 5.3: (Top) Positions in $\hat{x}$ are presented for three original demonstrations from the kinesthetic dataset. (Middle) Dynamic time warping (DTW) is applied to temporally align trials 1 and 2, which have the same number of cycles. (Bottom) DTW is applied to temporally align trials 1 and 3, which have different numbers of cycles. Note axis scales are the same for all three plots.

as illustrated in Figure 5.3. A second limitation with temporal methods is precisely their dependency on time, specifically when sharing control with humans or predicting human behavior. In these situations, task start times may be uncertain, and even if they are known, human actions do not always adhere to temporal dependency.

As a result, it is difficult to represent the data as a time-dependent trajectory, especially in the case of shared autonomy where a human exhibits actions that are variable in time. For this reason, a class of approaches focuses on learning autonomous *dynamical systems* (DS) to generate motion [**?**, **?**, **?**, **?**]. These approaches implicitly model temporal effects by

predicting the time-rate of change from system state, which is the approach taken here.

5.1.2 Multimodal dynamics

The second feature examined is the multimodal aspect of the data, in both motion and kinetics. Consider the velocity-position plots in in Figure 5.2. A popular dynamical system (DS) model form [**?**, **?**, **?**] assumes the velocity is generated by a Gaussian as a function of position

$$\dot{\boldsymbol{x}} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x})). \tag{5.2}$$

Equation (5.2) can be modeled using a variety of probabilistic regression techniques such a Gaussian Process Regression (GPR) [**?**] or Gaussian mixture regression (GMR) [**?**] among others[2]. However, this unimodal regression would result in an average $\boldsymbol{\mu} \approx \mathbf{0}$ for the demonstration data since there are both positive and negative motions at the same position $\boldsymbol{x}$. Similarly, examining the force-velocity plots shows that there is hysteresis in the brush during cleaning, resulting in differing forces depending on the stroke direction. A force-prediction model of the form

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}(\dot{\boldsymbol{x}}), \boldsymbol{\Sigma}(\dot{\boldsymbol{x}})) \tag{5.3}$$

would exhibit similar averaging issues as (5.2).

In general for dynamical tasks, conditioning on position and velocity $(\boldsymbol{x}, \dot{\boldsymbol{x}})$ is not guaranteed to yield unimodal predictions for both force $\boldsymbol{f}$ and acceleration $\ddot{\boldsymbol{x}}$ due to non-static motion behavior and nonlinear physics. For this reason, a method for isolating unimodal prediction regions is needed.

The challenge is how to isolate the data into sets of unimodal surfaces. At first glance, hierarchical clustering [**?**] seems well-suited for this problem. Clusters of similar data can be agglomerated into a tree data structure that defines a hierarchy of surfaces. However, arbitrary clustering in this fashion imposes no restrictions on how data moves through the clusters, which introduces additional issues when learning a time-evolving sequence. Further,

[2]For a comparison of regression techniques in the context of dynamics modeling, see [**?**].

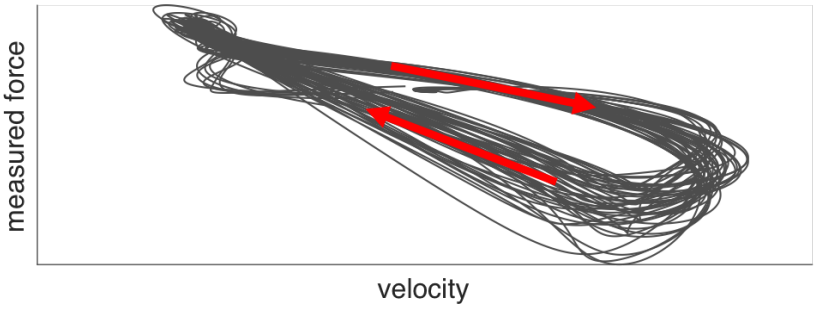


Figure 5.4: This work proposes a method to isolate unimodal predictions via the motion sequence. This figure shows two different force-prediction surfaces, with opposing motion direction indicated by the red arrows. By isolating the motion direction in this example, independent predictors for both directions can be found.

it is not clear what a good metric for "similar data" would mean in a motion sense. Because time evolution is a critical element of the model to be learned, this hierarchical clustering approach fails to produce a feasible model for the brush-cleaning dataset.

To address this issue, this chapter proposes a novel method to identify a set of latent actions from the motion sequence in order to find unimodal predictors. The basis of this concept is illustrated in Figure 5.4.

5.1.3 Motion bifurcations

The final feature examined is the bifurcation of motion (position and velocity) in both datasets. Consider the velocity-position plot in Fig. 5.5. The brush starts on the far left and begins moving with positive velocity toward the hole. As the brush stroke reaches its maxima at the far right, it begins a back stroke with negative velocity. The brush stroke repeats the positive and negative cleaning motions for several cycles with the brush remaining inserted in the pilot hole. However, at the end of the task, the brush exits the hole, moving back to the original starting location on the far left. This final exit motion diverges from negative stroke motion during cleaning, and presents a bifurcation in motion over.

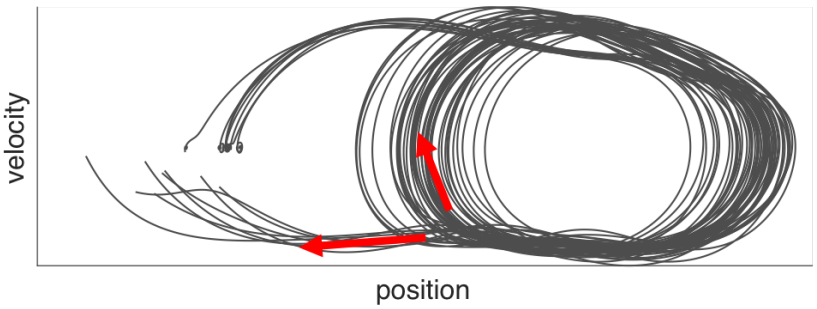


Figure 5.5: On the final cleaning stroke, the motion diverges from previous cycles to return to the starting condition. This presents a challenge when modeling the sequence of data.

One popular approach to model sequencing in imitation learning is to combine Gaussian mixture regression (GMR) [?] with Hidden Markov Models (HMM) [?] or, more-recently, hidden semi-Markov models (HSMM) [?, ?, ?]. In both approaches, the concept is to develop a generative model of the task using Gaussian Mixture Modeling (GMM) [?]. HMM or HSMM are used to model the sequential transitions of data between clusters in the GMM. At any given time, the current and next cluster are used in a GMR conditioning step to predict the variables of interest, e.g.: velocity [?, ?, ?, ?] or effort [?, ?]. Typically regressions are either normalized time t or position $\boldsymbol{x}$.

This two-cluster regression returns a localized regression surface, and has potential to address the challenge in Section 5.1.2. For this reason, the work in this chapter adopts the sequential GMM/GMR approach, however, there are several issues that arise when applying the common HMM methods to the brush-cleaning datasets.

1. Data variance introduces substantial noise into the transition sequence of data through a mixture model. This is a common problem in HMM, and is addressed through iterative path pruning to remove noise in the transition models. However, for the cleaning dataset, it is not clear how to rank paths for pruning. In many cases, noisy data may transition between two clusters several times before proceeding on, when in reality this is noise in the system. However, with a typical HMM pruning approach,

these adjacent transitions tend to receive high probabilities due to their frequency in the data and will be pruned after other—potentially more critical—paths in the sequence.

2. The Markov assumption breaks down in the brush-cleaning task. It is unclear how many previous clusters that the selection of the next cluster should be based on, therefore selection of the hidden state dimension in an HMM is difficult. This is particularly important in the last data bifurcation, where the data exits the cleaning cycle and returns to the initial position. This type of behavior occurs in many physical manufacturing operations (e.g. fastening, sanding, riveting), but is much less likely to be encountered in reaching operations. To address this issue, the proposed approach uses an autoregressive exogenous neural network (NARX) [**?**] to recreate the sequence of primitives. The external input provides indication to the network if the task is completed. Choice of these features influences the outcome of the generative sequence, and is thus a design consideration.

3. Conditioning on only two clusters at any one time places unnecessary burden on a sequencing algorithm. As the number of clusters increases, the sequence variance increases due to noisy data, causing degradation of sequence prediction performance for any choice of algorithm. However, while more clusters may better fit the data, there may a fixed optimal set of surfaces (collections of clusters) on which prediction should occur. This is precisely what the proposed approach seeks to find. Identifying such sets of clusters seeks to balance the model regression capability with the resulting sequence complexity.

To address these challenges, this work adopts the GMM/GMR framework, but develops different methods for both isolating regression surfaces and learning the sequence between them.

5.2 Preliminaries

5.2.1 Data collection

Data $\mathcal{D}$ is collected for several demonstrations either through kinesthetic or haptic teaching [?, ?]. The number of observations over all demonstrations is denoted as n_s. Collected signals include position $\boldsymbol{x}$, velocity $\boldsymbol{v}$, acceleration $\dot{\boldsymbol{v}}$, task efforts $\boldsymbol{f}$, and time t for samples $i = \{1, \ldots, n_s\}$, i.e.

$$\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{v}_i, \dot{\boldsymbol{v}}_i, \boldsymbol{f}_i, t_i\}_i^{n_s}. \tag{5.4}$$

The time is normalized such that $t = 0$ at the start of each demonstration. Several requirements are imposed on the demonstrations to aid in the learning and application:

- The data is assumed to be generated by an expert operator. This is important since the objective is to learn a good version of the task. In particular, an expert demonstration enables robotic assistance from learning to aide or outperform a novice operator.

- The data is assumed to be anomaly free. In the example brush-cleaning application, this means that the demonstration is i) free from unwanted contact of the brush with the work piece during insertion, and ii) free from the brush stopping mid-task due to friction since this is not the desired motion during the task. While these events may arise during some demonstrations, it is assumed that the demonstrator can remove these undesired demonstrations from the training set.

- The data is assumed to be collected under similar environmental conditions as the operating environment, i.e. in particular the task efforts $\boldsymbol{f}$ are consistent. In the brush-cleaning application, this requires the demonstration workpiece to be similar to the actual workpiece in practice. If a haptic rendering environment is used to produce a synthetic test bench for efficient data collection, the task efforts $\boldsymbol{f}$ must be sufficiently close to reality. Since this work examines a case where training occurs on the same

workpiece as that in practice, determining what is "sufficiently-close" for synthetic data generation is out of scope.

- At a minimum, three demonstrations are needed for cross-validation (two for learning first and second moments, and one for validation), although more is preferable to achieve a better estimate of demonstration variance. For the examined application, varying datasets of 7-15 demonstrations were used for the learning and analysis.

- The data is assumed to be filtered. In the data presented here, a zero-lag lowpass Butterworth filter with a passband of 10 Hz and a stopband of 30 Hz is used. This filter is used to remove aliasing due to the training sample rate of $\Delta = 1/200$ s. It is also used to reduce noise in the numerical derivative computation of acceleration $\dot{\boldsymbol{v}}$ from velocity $\boldsymbol{v}$. A passband of 10 Hz is selected since the human reaction time is known to be 0.1 s, and filtering frequencies less than this would remove elements of the human demonstration.

- The data is normalized such that $\mathbb{E}[\boldsymbol{x}] = \mathbf{0}$, i.e. the position data is centered around its first moment. This is conceptually useful for providing a consistent domain when localizing to new hole positions.

5.2.2 Task dynamics structure

Consider the simplified free-body diagram for the brush-cleaning task in Figure 5.6. In this system, a human operator exerts a force $\boldsymbol{u}$ onto the handle at position $\boldsymbol{x}$. In the case of kinesthetic demonstration, $\boldsymbol{u}$ is directly applied by the operator onto the tool handle, while in haptic demonstration, $\boldsymbol{u}$ is algorithmically applied onto the handle through a teleoperation scheme.

In addition to the human efforts, the workpiece exerts a force $\boldsymbol{f}$ onto the system during the cleaning motion. The force is assumed to be available at the position $\boldsymbol{x}$, i.e. through the kinematic relationship between the handle position and the sensor measurement position.

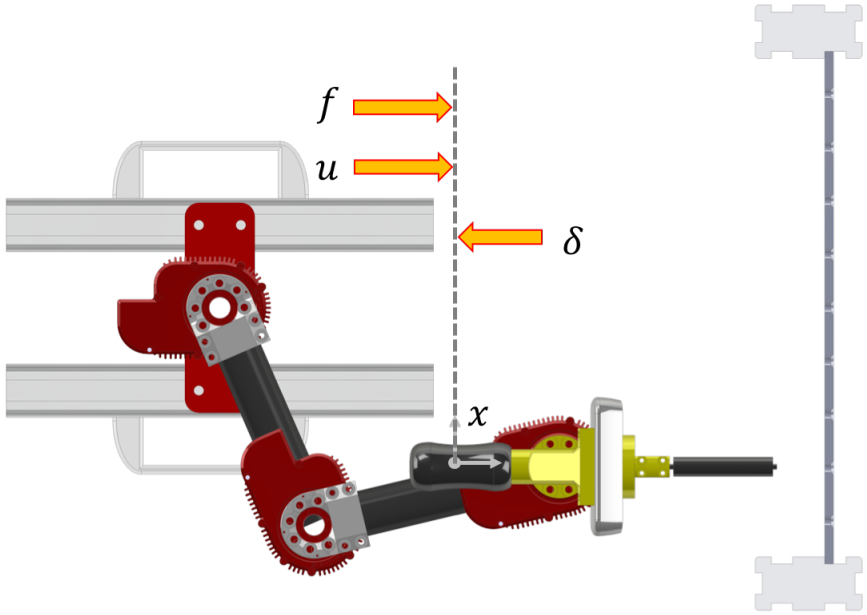


Figure 5.6: In the brush cleaning task, the human exerts a force $\boldsymbol{u}$ at the brush handle position $\boldsymbol{x}$. Forces $\boldsymbol{f}$ from the interaction of the brush with the workpiece are reflected back onto the handle. Additionally, uncompensated robot dynamics $\boldsymbol{\delta}$ are reflected onto the handle.

Finally, a collection of disturbance forces $\boldsymbol{\delta}$ are applied onto the handle. Examples of such disturbances include forces applied by a control law, or uncompensated robot dynamics. In the case of well-compensated force-controlled robots, this disturbance will be small. However, in this work, this disturbance is non-negligible due to the custom nature of the experimental platform.

Motion is defined by the acceleration $\dot{\boldsymbol{v}}$. Through Newton's laws of motion, the combined system and task dynamics take the second-order form

$$\dot{\boldsymbol{x}} = \boldsymbol{v} \tag{5.5}$$

$$\boldsymbol{\Lambda}\dot{\boldsymbol{v}} = \boldsymbol{f} + \boldsymbol{u} - \boldsymbol{\delta} \tag{5.6}$$

where $\boldsymbol{x}, \boldsymbol{v}, \dot{\boldsymbol{v}} \in \mathbb{R}^2$ are the position, velocity, and acceleration of the robot handle, $\boldsymbol{\Lambda} \in \mathbb{R}^{2\times2}$ is the operational space [**?**] inertia tensor matrix of the robotic system at the robot handle, and $\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{\delta} \in \mathbb{R}^2$ are the previously-mentioned efforts. Note that in general, these terms are not limited to $\in \mathbb{R}^2$, and can include 3D translation and rotation.

5.3 Model learning

5.3.1 Problem Statement

The imitation learning problem is to develop a control policy $\boldsymbol{u} = \pi(\boldsymbol{x}, \boldsymbol{v})$ that mimics the motion and exerted forces in the demonstration data $\mathcal{D}$. Taking a dynamical system approach results in the need to learn two function mappings of the form

$$\dot{\boldsymbol{v}} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{v}) \tag{5.7}$$

$$\boldsymbol{f} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{v}), \tag{5.8}$$

where $\boldsymbol{h}$ is an autonomous dynamical system and $\boldsymbol{g}$ is a force observation equation. These relationships are learned probabilistically, since uncertainty estimates are exploited in practice as in Chapters 3 and 4.

The learning problem is thus to find a model Ξ that maximizes the likelihood of the data $\mathcal{D}$, with some added penalty on the structure of Ξ, i.e.

$$\Xi^* = \arg\min_{\Xi} \; -\ln \mathcal{L}(\mathcal{D}|\Xi) + \lambda\|\Xi\|. \tag{5.9}$$

In this formulation, $\ln \mathcal{L}(\mathcal{D}|\Xi)$ is the log likelihood of the model Ξ for all training data $\mathcal{D}$ and λ is a regularization penalty, i.e. to promote better model generalization. A discussion on the regularization follows in Section 5.3.7.

Using the conditional model relationships in (5.7) and (5.8), the log likelihood of the data

$\mathcal{D}$ can be expanded to

$$\ln \mathcal{L}(\mathcal{D}|\Xi) = \sum_{i=1}^{n_s} \ln \{p(\mathcal{D}_i|\Xi)\} \tag{5.10}$$

$$= \sum_{i=1}^{n_s} \ln \{p(\boldsymbol{f}_i, \dot{\boldsymbol{v}}_i, \boldsymbol{v}_i, \boldsymbol{x}_i|\Xi)\} \tag{5.11}$$

$$= \sum_{i=1}^{n_s} \ln \{p(\boldsymbol{f}_i, \dot{\boldsymbol{v}}_i|\boldsymbol{v}_i, \boldsymbol{x}_i, \Xi) p(\boldsymbol{v}_i, \boldsymbol{x}_i|\Xi)\} \tag{5.12}$$

$$= \sum_{i=1}^{n_s} \ln \{p(\boldsymbol{f}_i, \dot{\boldsymbol{v}}_i|\boldsymbol{v}_i, \boldsymbol{x}_i, \Xi)\}\,. \tag{5.13}$$

Note that for the demonstration data, $p(\boldsymbol{v}_i, \boldsymbol{x}_i|\Xi) = 1$.

It will be useful to define an augmented system state $\boldsymbol{\chi} = (\boldsymbol{x}, \boldsymbol{v})^\top$ and its time-rate of change $\dot{\boldsymbol{\chi}} = (\dot{\boldsymbol{x}}, \dot{\boldsymbol{v}})^\top$ in a control sense. Note that the velocity relationship $\dot{\boldsymbol{x}} = \boldsymbol{v}$ is known exactly, but the acceleration prediction in (5.7), i.e. $\dot{\boldsymbol{v}} = \boldsymbol{h}(\boldsymbol{\chi})$ must be learned.

5.3.2 Solution approach

To address the challenges discussed in Section 5.1, this work proposes a state-action decomposition of the clustered data. This concept addresses the previous issues as illustrated in Figure 5.7. By carefully selecting clusters in transitional regions, i.e. a set of states[3] $\mathcal{S}$, clusters in between these regions can be isolated as unimodal prediction surfaces, i.e. a set of actions $\mathcal{A}$. Under this model, equations (5.7) and (5.8) are now modified to become

$$\dot{\boldsymbol{v}} = \boldsymbol{h}(\boldsymbol{\chi}; a) + \boldsymbol{\nu}, \quad \boldsymbol{\nu} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}(\boldsymbol{\chi}; a)) \tag{5.14}$$

$$\boldsymbol{f} = \boldsymbol{g}(\boldsymbol{\chi}; a) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}(\boldsymbol{\chi}; a)) \tag{5.15}$$

with the addition of a dynamic process q on the evolution of action a

$$a_{i+1} = q(\boldsymbol{\chi}_i; a_i) \tag{5.16}$$

where i is the current time step.

[3]Note that "state" will be used when discussing the subset of clusters $s \in \mathcal{S}$, and "system state" will be used when discussing the vector $\boldsymbol{\chi}$.

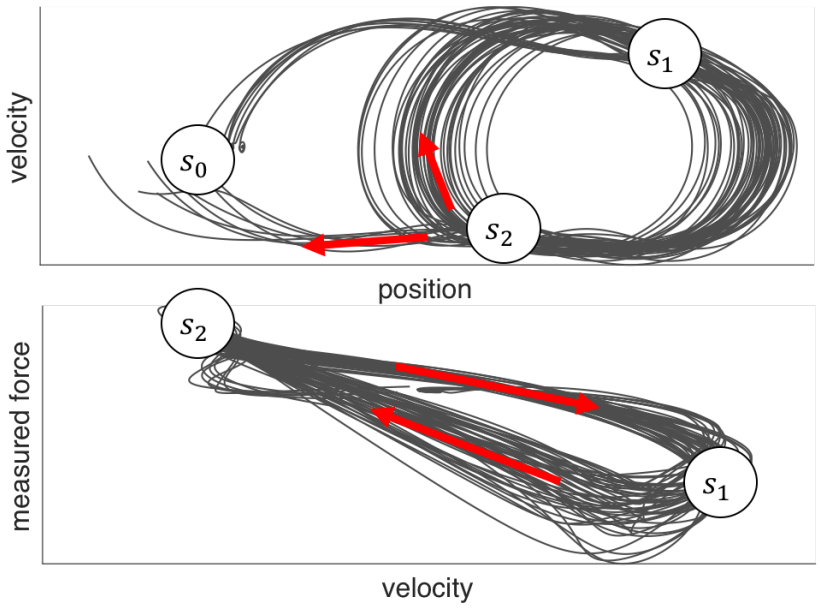


Figure 5.7: The proposed state/action decomposition addresses both the unimodal prediction problem and sequencing issue. By identifying transitional states $\{s_0, s_1, s_2\}$, regions in between these states (i.e. actions) can be isolated for unimodal regression.

With the independence assumptions of (5.14) (5.15), and (5.16) depicted in the graphical model in Figure 5.8, the model likelihood in (5.13) can be expanded to include actions $a \in \mathcal{A}$

$$\ln \mathcal{L}(\mathcal{D}|\Xi) = \sum_{i=1}^{n_s} \ln \{p(\boldsymbol{f}_i, \dot{\boldsymbol{v}}_i|\boldsymbol{\chi}_i, \Xi)\} \tag{5.17}$$

$$= \sum_{i=1}^{n_s} \ln \left\{ \sum_{a_i \in \mathcal{A}} p(\boldsymbol{f}_i, \dot{\boldsymbol{v}}_i|\boldsymbol{\chi}_i, a_i, \Xi) p(a_i) \right\} \tag{5.18}$$

$$= \sum_{i=2}^{n_s} \ln \left\{ \sum_{a_i \in \mathcal{A}} p(\boldsymbol{f}_i, \dot{\boldsymbol{v}}_i|\boldsymbol{\chi}_i, a_i, \Xi) \sum_{a_{i-1} \in \mathcal{A}} p(a_i|a_{i-1}, \boldsymbol{\chi}_{i-1}, \Xi) p(a_{i-1}) \right\}. \tag{5.19}$$

There are three elements to the modeling approach, consisting of (i) data clustering and regression, (ii) state-action decomposition, and (iii) action sequence learning. These are explored in the following subsections.

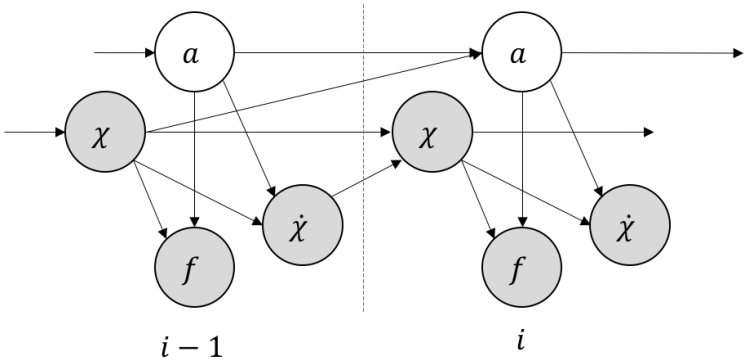


Figure 5.8: A graphical model (Dynamic Bayesian network) depicts the conditional dependency of signals at the current time step i and the previous time step $i-1$.

5.3.3 Clustering and Gaussian mixture regression (GMR)

Clustering. Define a reduced dataset $\bar{\mathcal{D}} = \{\boldsymbol{\chi}_i, \dot{\boldsymbol{v}}_i, \boldsymbol{f}_i\}_{i=1}^{n_s}$, i.e. one that omits time from the original dataset in (5.4). Data $\bar{\mathcal{D}}$ is clustered into a set of clustered states $c \in \mathcal{C}$ using the Gaussian mixture model (GMM) [**?**]. The GMM assumes the form

$$p(\bar{\mathcal{D}}) = \sum_{k=1}^{K} \omega_k \mathcal{N}(\bar{\mathcal{D}} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{5.20}$$

where ω are individual component proportions, $\sum \omega = 1$, and K is the number of clusters. The k^{th} cluster c_k is defined as the parameter set $c_k = \{\omega, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}_k$, where ω are weights. The expanded data mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ for cluster c are

$$\boldsymbol{\mu}_c = \begin{pmatrix} \boldsymbol{\mu}_{\chi,c} \\ \boldsymbol{\mu}_{\dot{v},c} \\ \boldsymbol{\mu}_{f,c} \end{pmatrix}, \quad \boldsymbol{\Sigma}_c = \begin{pmatrix} \boldsymbol{\Sigma}_{\chi,c} & \boldsymbol{\Sigma}_{\chi\dot{v},c} & \boldsymbol{\Sigma}_{\chi f,c} \\ \boldsymbol{\Sigma}_{\dot{v}\chi,c} & \boldsymbol{\Sigma}_{\dot{v},c} & \boldsymbol{\Sigma}_{\dot{v}f,c} \\ \boldsymbol{\Sigma}_{f\chi,c} & \boldsymbol{\Sigma}_{f\dot{v},c} & \boldsymbol{\Sigma}_{f,c} \end{pmatrix} \tag{5.21}$$

where diagonal elements of $\boldsymbol{\Sigma}$ are individual signal variance in the cluster, and off-diagonal elements correlate signals.

Equation (5.20) is learned using the EM algorithm [**?**], initialized via k-means. Note that k-means uses a stochastic initialization, therefore in practice several replicates of the

algorithm are attempted and the best fit of these models is selected. This data clustering approach requires selection of the number of clusters K, which is addressed during model selection in Section 5.3.7.

Regression. To predict the acceleration $\dot{\boldsymbol{v}}$ and force $\boldsymbol{f}$ in (5.14) and (5.15) using a subset of clusters to define action $a \subset \mathcal{C}$, conditioning is applied to the GMM in (5.20). This is accomplished by taking the axiom of conditional probability

$$p(Y|X) = \frac{p(Y,X)}{p(X)}$$

to compute some output space Y as a function of some input space X, where $p(Y,X)$ and $p(X)$ are elements of the generative Gaussian mixture model. For the bivariate Gaussian model

$$p(X,Y) \sim \mathcal{N}\left(\begin{bmatrix}\mu_X\\ \mu_Y\end{bmatrix}, \begin{bmatrix}\Sigma_X & \Sigma_{XY}\\ \Sigma_{YX} & \Sigma_Y\end{bmatrix}\right),$$

Gaussian conditioning [**?**] yields

$$p(Y|X) \sim \mathcal{N}(m(X), \sigma^2) \tag{5.22}$$

where

$$m(X) = \mu_Y + \Sigma_{YX}\Sigma_X^{-1}(X - \mu_X)$$
$$\sigma^2 = \Sigma_Y - \Sigma_{YX}\Sigma_X^{-1}\Sigma_{XY}.$$

This result can be applied to the GMM in (5.20). Letting $X = \boldsymbol{\chi},\ Y = \dot{\boldsymbol{v}}$, the acceleration expectation in (5.14) expands to

$$\mathbb{E}[\dot{\boldsymbol{v}}|\boldsymbol{\chi}] = \boldsymbol{h}(\boldsymbol{\chi}; a) = \sum_{c\in a} \omega_c(\boldsymbol{\chi})\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi}), \tag{5.23}$$

where $\omega_c(\boldsymbol{\chi})$ is a local weighting scalar and $\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi})$ is the local acceleration mean conditioned on system state $\boldsymbol{\chi}$. Similarly, letting $X = \boldsymbol{\chi},\ Y = \boldsymbol{f}$, the force expectation in (5.15) is

$$\mathbb{E}[\boldsymbol{f}|\boldsymbol{\chi}] = \boldsymbol{g}(\boldsymbol{\chi}; a) = \sum_{c\in a} \omega_c(\boldsymbol{\chi})\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi}). \tag{5.24}$$

where $\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi})$ is the local force mean conditioned on $\boldsymbol{\chi}$. From the Gaussian conditioning result in (5.22), the local means $\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi})$ and $\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi})$, i.e. for each cluster c, are

$$\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi}) = \boldsymbol{\mu}_{\dot{v},c} + \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}(\boldsymbol{\chi} - \boldsymbol{\mu}_{\chi,c}) \tag{5.25}$$

$$\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi}) = \boldsymbol{\mu}_{f,c} + \boldsymbol{\Sigma}_{f\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}(\boldsymbol{\chi} - \boldsymbol{\mu}_{\chi,c}), \tag{5.26}$$

where the local system state covariance $\boldsymbol{\Sigma}_{\chi,c} \in \mathbb{R}^{4\times4}$, acceleration covariance $\boldsymbol{\Sigma}_{\dot{v}\chi,c} \in \mathbb{R}^{2\times4}$, force covariance $\boldsymbol{\Sigma}_{f\chi,c} \in \mathbb{R}^{2\times4}$, acceleration mean $\boldsymbol{\mu}_{\dot{v},c} \in \mathbb{R}^2$, and force mean $\boldsymbol{\mu}_{f,c} \in \mathbb{R}^2$ are the cluster parameters in (5.21).

The acceleration covariance in (5.14) expands to

$$\begin{aligned}\boldsymbol{Q}(\boldsymbol{\chi}; a) &= \mathbb{E}\left[(\dot{\boldsymbol{v}} - \mathbb{E}[\dot{\boldsymbol{v}}])\,(\dot{\boldsymbol{v}} - \mathbb{E}[\dot{\boldsymbol{v}}])^\top\right] \\ &= \sum_{c\in a}\omega_c(\boldsymbol{\chi})\left(\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi})^2 + \breve{\boldsymbol{\Sigma}}_{\dot{v},c}\right) - \left(\sum_{c\in a}\omega_c(\boldsymbol{\chi})\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi})\right)^2,\end{aligned} \tag{5.27}$$

where $\breve{\boldsymbol{\Sigma}}_{\dot{v},c}$ is the local acceleration covariance from conditioning. Similarly, the force covariance in (5.15) is

$$\begin{aligned}\boldsymbol{R}(\boldsymbol{\chi}; a) &= \mathbb{E}\left[(\boldsymbol{f} - \mathbb{E}[\boldsymbol{f}])\,(\boldsymbol{f} - \mathbb{E}[\boldsymbol{f}])^\top\right] \\ &= \sum_{c\in a}\omega_c(\boldsymbol{\chi})\left(\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi})^2 + \breve{\boldsymbol{\Sigma}}_{f,c}\right) - \left(\sum_{c\in a}\omega_c(\boldsymbol{\chi})\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi})\right)^2.\end{aligned} \tag{5.28}$$

where $\breve{\boldsymbol{\Sigma}}_{f,c}$ is the local force covariance from conditioning. From the Gaussian conditioning result in (5.22), the local covariances $\breve{\boldsymbol{\Sigma}}_{\dot{v},c}$ and $\breve{\boldsymbol{\Sigma}}_{f,c}$, i.e. for each cluster c, are

$$\breve{\boldsymbol{\Sigma}}_{\dot{v},c} = \boldsymbol{\Sigma}_{\dot{v},c} - \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}\boldsymbol{\Sigma}_{\chi\dot{v},c} \tag{5.29}$$

$$\breve{\boldsymbol{\Sigma}}_{f,c} = \boldsymbol{\Sigma}_{f,c} - \boldsymbol{\Sigma}_{f\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}\boldsymbol{\Sigma}_{\chi f,c}, \tag{5.30}$$

where each covariance is from the cluster parameters in (5.21).

Finally, the weighting component mixes the local models based on the normalized system state $\boldsymbol{\chi}$ posterior

$$\omega_c(\boldsymbol{\chi}) = \frac{\omega_c\mathcal{N}(\boldsymbol{\chi}|\boldsymbol{\mu}_{\chi,c}, \boldsymbol{\Sigma}_{\chi,c})}{\sum_{a\in c}\omega_c\mathcal{N}(\boldsymbol{\chi}|\boldsymbol{\mu}_{\chi,c}, \boldsymbol{\Sigma}_{\chi,c})}. \tag{5.31}$$

For a full derivation of the Gaussian mixture regression, see [**?**].

Linear model form. The local acceleration expectation in (5.25) can be rearranged to assume a linear form

$$\breve{\boldsymbol{\mu}}_{\dot{v},c}(\boldsymbol{\chi}) = \boldsymbol{\mu}_{\dot{v},c} + \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}(\boldsymbol{\chi} - \boldsymbol{\mu}_{\chi,c}) \tag{5.32}$$

$$= \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}\boldsymbol{\chi} + \boldsymbol{\mu}_{\dot{v},c} - \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}\boldsymbol{\mu}_{\chi,c} \tag{5.33}$$

$$= \bar{\boldsymbol{A}}_c\boldsymbol{\chi} + \bar{\boldsymbol{b}}_c \tag{5.34}$$

where

$$\bar{\boldsymbol{A}}_c \in \mathbb{R}^{2\times 4} = \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1} \tag{5.35}$$

$$\bar{\boldsymbol{b}}_c \in \mathbb{R}^{2} = \boldsymbol{\mu}_{\dot{v},c} - \boldsymbol{\Sigma}_{\dot{v}\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}\boldsymbol{\mu}_{\chi,c}. \tag{5.36}$$

Similarly, the force expectation in (5.26) can be arranged

$$\breve{\boldsymbol{\mu}}_{f,c}(\boldsymbol{\chi}) = \bar{\boldsymbol{C}}_c\boldsymbol{\chi} + \bar{\boldsymbol{d}}_c \tag{5.37}$$

where

$$\bar{\boldsymbol{C}}_c \in \mathbb{R}^{2\times 4} = \boldsymbol{\Sigma}_{f\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1} \tag{5.38}$$

$$\bar{\boldsymbol{d}}_c \in \mathbb{R}^{2} = \boldsymbol{\mu}_{f,c} - \boldsymbol{\Sigma}_{f\chi,c}\boldsymbol{\Sigma}_{\chi,c}^{-1}\boldsymbol{\mu}_{\chi,c}. \tag{5.39}$$

Let $\bar{\boldsymbol{A}}_{c,l} \in \mathbb{R}^{2\times 2}$ denote the left block of $\bar{\boldsymbol{A}}_c$, and $\bar{\boldsymbol{A}}_{c,r} \in \mathbb{R}^{2\times 2}$ denote the right block of $\bar{\boldsymbol{A}}_c$. Also let $\bar{\boldsymbol{C}}_{c,l} \in \mathbb{R}^{2\times 2}$ denote the left block of $\bar{\boldsymbol{C}}_c$, and $\bar{\boldsymbol{C}}_{c,r} \in \mathbb{R}^{2\times 2}$ denote the right block of $\bar{\boldsymbol{C}}_c$. Then the linear system of equations for each cluster can be expressed as

$$\begin{pmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{v}} \\ \boldsymbol{f} \end{pmatrix} = \left(\begin{array}{cc|c} \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \bar{\boldsymbol{A}}_{c,l} & \bar{\boldsymbol{A}}_{c,r} & \bar{\boldsymbol{b}}_c \\ \hline \bar{\boldsymbol{C}}_{c,l} & \bar{\boldsymbol{C}}_{c,r} & \bar{\boldsymbol{d}}_c \end{array}\right) \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{v} \\ \boldsymbol{1} \end{pmatrix} = \left(\begin{array}{c|c} \boldsymbol{A}_c & \boldsymbol{B}_c \\ \hline \boldsymbol{C}_c & \boldsymbol{D}_c \end{array}\right) \begin{pmatrix} \boldsymbol{\chi} \\ \boldsymbol{1} \end{pmatrix}. \tag{5.40}$$

For a set of clusters $a \subset \mathcal{C}$, the complete system dynamics is

$$\begin{aligned} \dot{\boldsymbol{\chi}} &= \boldsymbol{A}(\boldsymbol{\chi}, a)\boldsymbol{\chi} + \boldsymbol{B}(\boldsymbol{\chi}, a) \\ \boldsymbol{f} &= \boldsymbol{C}(\boldsymbol{\chi}, a)\boldsymbol{\chi} + \boldsymbol{D}(\boldsymbol{\chi}, a) \end{aligned} \tag{5.41}$$

where, from (5.23) (5.24), and (5.40)

$$\boldsymbol{A}(\boldsymbol{\chi}, a) = \sum_{c \in a} \omega_c(\boldsymbol{\chi}) \boldsymbol{A}_c \tag{5.42}$$

$$\boldsymbol{B}(\boldsymbol{\chi}, a) = \sum_{c \in a} \omega_c(\boldsymbol{\chi}) \boldsymbol{B}_c \tag{5.43}$$

$$\boldsymbol{C}(\boldsymbol{\chi}, a) = \sum_{c \in a} \omega_c(\boldsymbol{\chi}) \boldsymbol{C}_c \tag{5.44}$$

$$\boldsymbol{D}(\boldsymbol{\chi}, a) = \sum_{c \in a} \omega_c(\boldsymbol{\chi}) \boldsymbol{D}_c. \tag{5.45}$$

This shows that the GMR for the desired task dynamics model can be conveniently represented as a locally-weighted linear state space model, which has useful properties for estimation and control [**?**].

5.3.4 Finding candidate states and actions

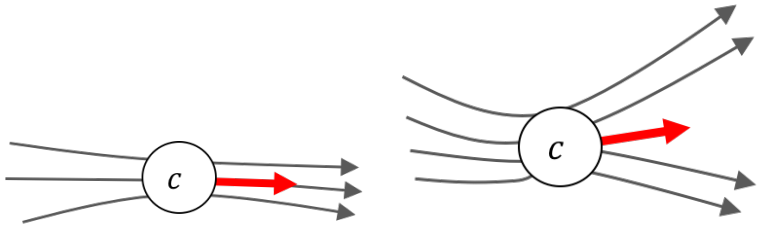
In order for the model in (5.41) to be a good prediction, a set of actions $a \in \mathcal{A}$ need to be identified. There are two requirements imposed on this action set. First, a well-chosen action will be isolated to unimodal output data over the input space because the underlying models are linear and Gaussian. Second, a good choice of actions will exhibit a consistent sequence, i.e. due to the dynamic process in (5.16).

This is the core problem addressed by the state-action decomposition, of which there are two elements to the solution. First, a set of clusters are isolated as states, i.e. $\mathcal{S} \subseteq \mathcal{C}$, defining boundaries between actions. Second, using this set of states $\mathcal{S}$, sets of clusters $\mathcal{A} = \{a \subset \mathcal{C}\}$ are identified in order to provide a prediction of motion and force in between the states.

Candidate states. One state identification approach looks for equilibrium points $\boldsymbol{\chi}^*$ in the motion data where

$$\dot{\boldsymbol{\chi}} = \boldsymbol{h}(\boldsymbol{\chi}^*, a) = \boldsymbol{0}. \tag{5.46}$$

However, these points do not exist for the brush cleaning data since the brush never stops moving during the cleaning action. Instead, the proposed method for isolating candidate



(a) Low divergence (b) High divergence

Figure 5.9: Identification of state candidates is accomplished by identifying clusters with high local divergence in motion.

states is to rank local regions of high motion divergence. The concept is illustrated in Figure 5.9. The motivation for this choice is that regions of low divergence, as shown in Figure 5.9a are more likely to be intermediate regions between a single set of boundary clusters, and that regions of high divergence are more likely to represent the boundaries, shown in Figure 5.9b.

Low divergence in local motion can be represented by streamline similarity, i.e. consistency of data local to a cluster to match the motion captured by the cluster mean. This rank r_c for each cluster c can be quantified by the mean dot product

$$r_c = \frac{1}{n_c} \sum_{j \in c} \dot{\boldsymbol{\chi}}_j \cdot \dot{\boldsymbol{\chi}}_c \tag{5.47}$$

where $\dot{\boldsymbol{\chi}}$ is the system state derivative, $j \in c$ denotes data proximal to cluster c consisting of n_c data points, and $\dot{\boldsymbol{\chi}}_c$ is the mean system state derivative defined by cluster c. A lower r_c indicates dissimilar streamlines, i.e. higher divergence, and thus a good candidate state.

Given this measure of similarity, N states can be selected from the lowest ordered values of r_c. An example of clusters selected as states via this ranking is shown in Figure 5.10. While a certain number of states N may be desired for specific applications, in general this value is unknown. Selection of this parameter is addressed during model selection in Section 5.3.7.

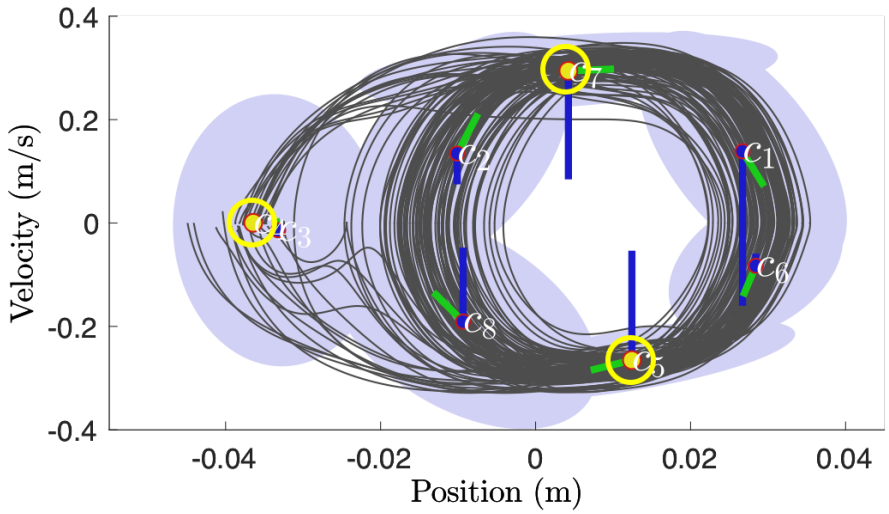


Figure 5.10: For a model with $K = 8$ clusters and $N = 3$ states, the set of state candidates $\mathcal{S} = \{c_3, c_5, c_7\}$ (circled in yellow). Gray lines indicate training data. For each cluster c, shaded regions indicate the covariance in system state $\boldsymbol{\Sigma}_{\chi,c}$, green arrows indicate mean acceleration $\boldsymbol{\mu}_{\dot{v},c}$ and blue arrows indicate mean force $\boldsymbol{\mu}_{f,c}$.

Action sets. Once the states $\mathcal{S}$ are identified, actions $a \in \mathcal{A}$ can be found by keeping track of the clusters visited in between the states. This can be accomplished by collecting unique paths between clusters through the stochastic adjacency matrix $\boldsymbol{W}_c$. An example adjacency representation is shown in Figure 5.11. This matrix represents the connectivity of clusters in a directed graph, and predicts the probabilistic forward movement through clusters

$$\boldsymbol{p}(c_i|c_{i-1}) = \boldsymbol{W}_c \boldsymbol{p}(c_{i-1}) \tag{5.48}$$

where $\boldsymbol{p}(c_i|c_{i-1})$ is the probability of moving to cluster c_i from the previous cluster c_{i-1}. Unique paths are found using breadth-first search between the states $s \in \mathcal{S}$ from the previous section. Algorithm 1 presents this search algorithm. These paths encode actions that traverse the clusters between states. Figure 5.12 shows an example set of learned actions.

The adjacency matrix $\boldsymbol{W}_c$ can be found from the following. Define $\boldsymbol{\Psi}_i = [\psi_2, \ldots, \psi_{n_s}]$ and $\boldsymbol{\Psi}_{i-1} = [\psi_1, \ldots, \psi_{n_s-1}]$ to be matrix collections of snapshots of one-hot cluster classification

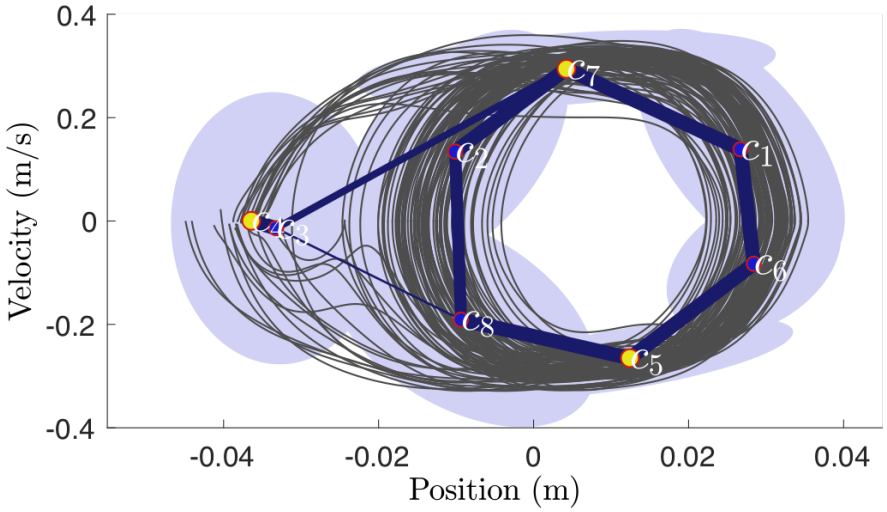


Figure 5.11: For a model with $K = 8$ clusters, blue lines indicate the cluster connectivity contained in the stochastic adjacency matrix $\boldsymbol{W}_c$. Gray lines indicate training data. Shaded regions indicate the covariance in system state $\boldsymbol{\Sigma}_{\chi,c}$ for each cluster c.

vectors $\psi_i = \text{GMM}(\bar{\mathcal{D}}_i)$[4]. The stochastic matrix $\boldsymbol{M}_c$ is found from the minimization

$$\min \ \|\boldsymbol{\Psi}_i - \boldsymbol{M}_c \boldsymbol{\Psi}_{i-1}\|_F^2 \tag{5.49}$$

where $\|\cdot\|_F$ is the matrix Frobenius norm. The stochastic adjacency matrix $\boldsymbol{W}_c$ is then

$$\boldsymbol{W}_c = \boldsymbol{M}_c - \text{diag}(\boldsymbol{M}_c) \tag{5.50}$$

where $\text{diag}(\cdot)$ denotes the matrix of diagonal elements.

Remark 4 *Consider the clusters $\mathcal{C}$, states $\mathcal{S}$, and actions $\mathcal{A}$ shown in Figure 5.13. Each action constitutes a static regression surface, defined by a set of clusters. This can be interpreted as a set of static task dynamics primitives, containing the desired motion and expected forces during each primitive. The phase plane of these motion primitives can be visualized by*

[4]A one-hot vector is a common representation for the result of a classification operation as a $K \times 1$ vector with 1 at the identified class index $k \in \{1, \ldots, K\}$ and 0 everywhere else.

Algorithm 1 Find actions using breadth-first search (BFS)

Input: $\boldsymbol{W}_c$ (stochastic adjacency), $\mathcal{S}$ (states)

Output: $\mathcal{A}$ (actions)

function $\mathcal{A}$ = FindActions($\boldsymbol{W}_c, \mathcal{S}$)
  `open_paths` $\leftarrow \{\mathcal{S}\}$ // Set up FIFO queue
  `closed_paths` $\leftarrow \emptyset$ // Empty set
  while $\neg$ **empty**(`open_paths`) **do**
    `path` $\leftarrow$ **denqueue**(`open_paths`) // A path is a set of clusters $\{c\}$
    $c_i \leftarrow$ **getLastCluster**(`path`) // Get the last cluster from the path
    $p(c_{i+1}) \leftarrow$ **simulateForward**(c_i) // From (5.48)
    for Each $c_{i+1} \in p(c_{i+1}) > 0$ **do** // For each adjacent cluster
      if $c_{i+1} \notin$ `path` **then** // If unvisited in the path
        `path` $\leftarrow$ **pushBack**(c_{i+1}) // Add state to path
        if $c_{i+1} \in \mathcal{S}$ **then** // A state has been reached, close
          `closed_paths` $\leftarrow$ **pushBack**(`path`)
        else // A state has not yet been reached, keep open
          `open_paths` $\leftarrow$ **enqueue**(`path`)
        end if
      end if
    end for
  end while
  $\mathcal{A} \leftarrow$ **mergeSimilarPaths**(`closed_paths`) // Similar means same first/last state
  return $\mathcal{A}$
end function

the vector field of system state derivatives $\dot{\boldsymbol{\chi}}$ over the system state $\boldsymbol{\chi}$. Note that the vector fields are only well-behaved in the regions near the clusters in the action set $c \subset a$. Outside of these regions, the vector field is nonlinear due to the local weighting of the Gaussian mixture

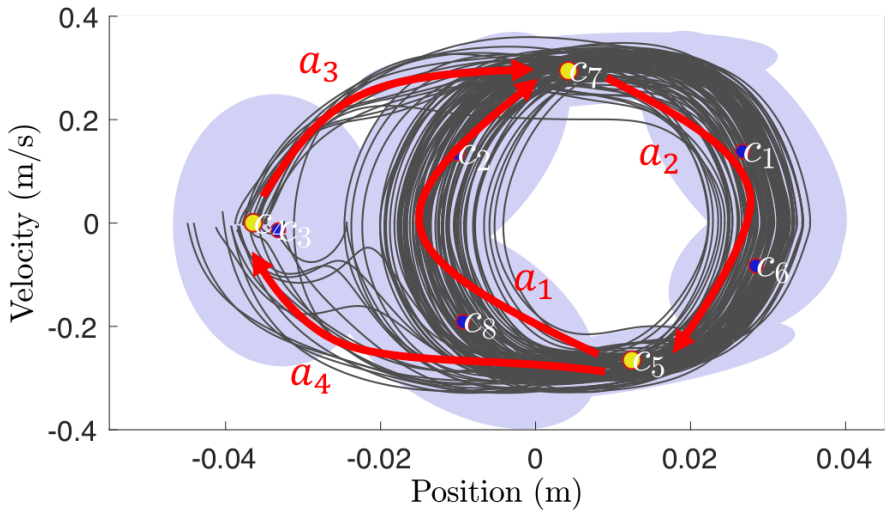


Figure 5.12: For a model with $K = 8$ clusters and $N = 3$ states, the set of actions $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ traverses the clusters via the red arrows. Gray lines indicate training data. Shaded regions indicate the covariance in system state $\mathbf{\Sigma}_{\chi,c}$ for each cluster c.

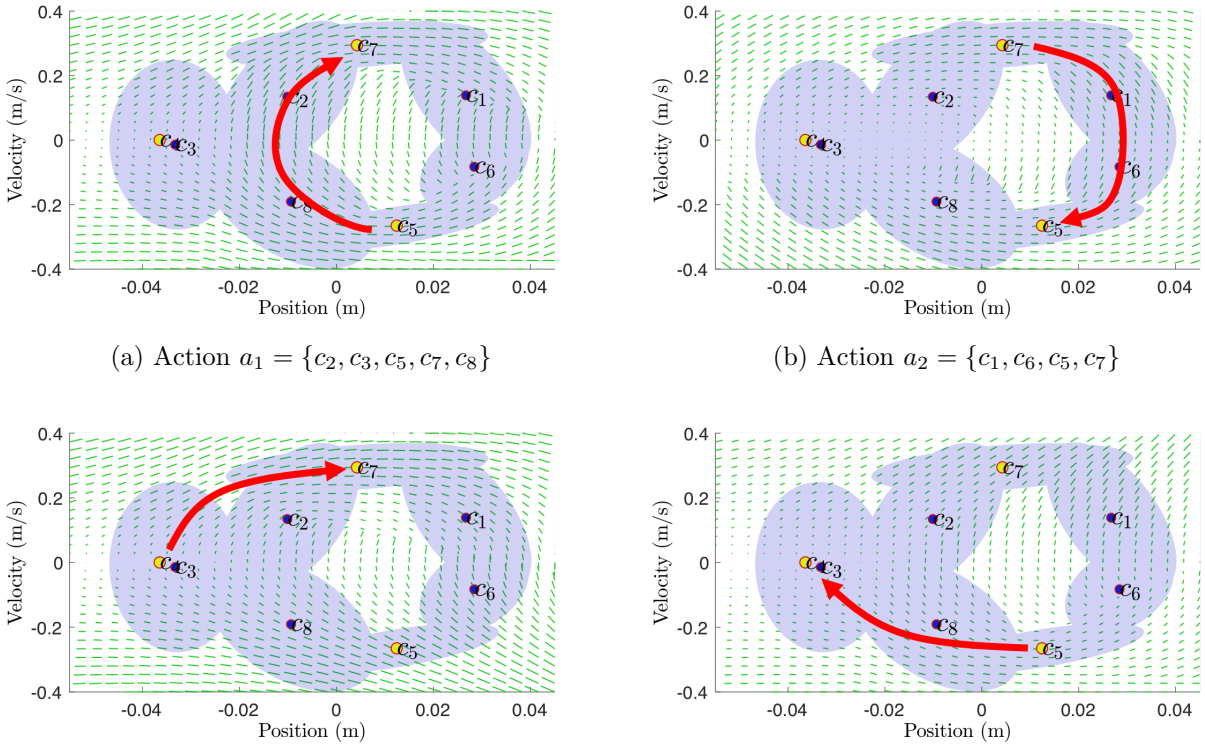
regression.

Some clusters are included in multiple actions. For example, cluster c_8 is included in actions a_1 and a_4. This is because trajectories pass through this region during each of these two primitives, therefore the local model is included in both regression surfaces.

5.3.5 Learning the action sequence

The remaining step is to learn the action sequence, which produces the model $p(a_{i+1}|a_i, \boldsymbol{\xi}_i)$ in (5.16) and (5.19). This model predicts the next action a_{i+1} from current action a_i, and a feature vector $\boldsymbol{\xi}_i$. Because actions a are uniquely identifiable by starting and ending states s, it is only necessary to learn a model that predicts adjacent transitions (i.e. from the current action to a new action) rather than also needing to find when the transitions occur, allowing for standard approaches to be used for the sequence prediction.

Sequencing will be determined by examining the following: (i) pre-processing to isolate



(a) Action $a_1 = \{c_2, c_3, c_5, c_7, c_8\}$

(b) Action $a_2 = \{c_1, c_6, c_5, c_7\}$

(c) Action $a_3 = \{c_4, c_3, c_7\}$

(d) Action $a_4 = \{c_3, c_4, c_5, c_8\}$

Figure 5.13: Four actions are identified for $K = 8$ clusters and $N = 3$ states. Each action predicts a motion vector field (shown in green arrows), i.e. the system state derivative as a function of system state $\boldsymbol{\chi} \rightarrow \dot{\boldsymbol{\chi}}$. The general motion contained in the data is depicted by the red arrows for each action. States are denoted in yellow.

the actions in the data, (ii) selecting exogenous features, and (iii) learning the adjacent transitions with a nonlinear autoregressive exogenous (NARX) model.

Isolating action transitions in the dataset. Actions $\mathcal{A}$ are identified in the dataset by finding the peak likelihood within activated states $\mathcal{S}$[5]. These peaks are the transitional points between actions, and the actions are uniquely determined by the starting and ending

[5]Let "activated" states refer to regions in the data that have been identified via clustering

states, based on how $\mathcal{A}$ is found in Section 5.3.4.

State activations $\mathcal{S} \subset \mathcal{C}$ are found by clustering the data $\bar{\mathcal{D}}$ using (5.20), and are isolated from this dataset. For each activation, the sample index corresponding to the minimum Mahalanobis distance M_D for activated state s determines the peak likelihood, and thus the index where transitions occur between actions, where the Mahalanobis distance is

$$M_D(\bar{\mathcal{D}}_i, s) = \left(\left(\bar{\mathcal{D}}_i - \boldsymbol{\mu}_s \right)^\top \boldsymbol{\Sigma}_s^{-1} \left(\bar{\mathcal{D}}_i - \boldsymbol{\mu}_s \right) \right)^{1/2}. \tag{5.51}$$

Actions are uniquely identified by the starting and ending states s for the segment. Having found the actions, define an augmented dataset $\breve{\mathcal{D}}$ to facilitate the sequence learning

$$\breve{\mathcal{D}} = \{ \boldsymbol{\chi}_i, \dot{\boldsymbol{v}}_i, \boldsymbol{f}_i, a_i, a_{i+1} \}_{i=1}^{n_s - 1}. \tag{5.52}$$

Exogenous feature selection. Exogenous features must be selected for the model inputs. For the cleaning task, these features primarily determine when the task is completed, since the autoregressive nature of the model is sufficient to capture Markov-conforming transitions. Ideally, a feature indicating hole cleanliness would be used to determine when the task is completed. In this hypothetical feature, each brush stroke would remove debris and grime from the hole, leaving the hole cleaner than before the stroke. Cast as a Markov Decision Process (MDP), each stroke constitutes accumulating a reward. Once most of the debris has been removed however, continuing to clean accumulates less and less reward, when at some point, the robot should stop cleaning since the task is complete. Unfortunately, hole cleanliness is not directly observable for the system and a sizable computer vision effort would be needed to produce this information. However, this concept is useful for selecting proxy features for determining task completion.

From the above example, it is important that the task completion features are integrators to represent a notion of "accumulation" during the task. Generally speaking, appropriate features will vary from task to task. In many instances, the operator may want to specify these features explicitly. If not, a set of features can be incorporated into the model and sparsity-promoting regularization added during learning to identify appropriate features

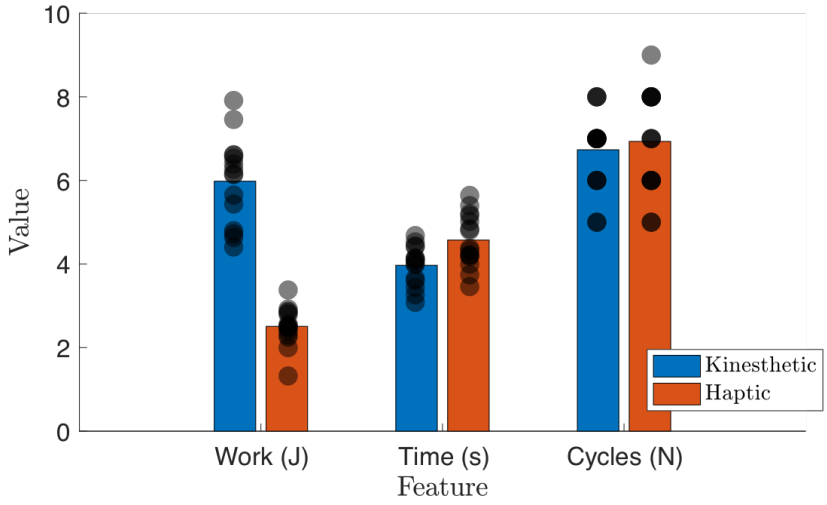


Figure 5.14: Several features–work, time, and number of cycles–are compared for both the kinesthetic and haptic datasets.

from the set. This chapter consider three features that exhibit the accumulation property: (i) accumulated task time, (ii) accumulated number of cycles, and (iii) mechanical work. Note that these features do not exhibit the *diminishing return* that is expected in the preceding example. In an MDP sense, this means that cleaning indefinitely would continue to accumulate reward, therefore these are not useful features for an MDP reward function.

Task time t is directly available in the training dataset. Accumulated number of cycles can be examined by accumulating one of the stroke actions in the set $\mathcal{A}$. Mechanical work can be determined from the integral

$$W(t) = \int_0^t \boldsymbol{f}(\tau)^\top \boldsymbol{v}(\tau)\, d\tau. \tag{5.53}$$

The distribution of these features for both the kinesthetic and haptic dataset is shown in Figure 5.14. Note the haptic dataset exhibits both lower forces and slower speeds than the kinesthetic dataset due to limitations of the haptic teleoperation. For this reason, the work feature varies greatly between the two datasets. Time is fairly consistent between the two datasets. The number of stroke cycles is also fairly consistent between the datasets. Note

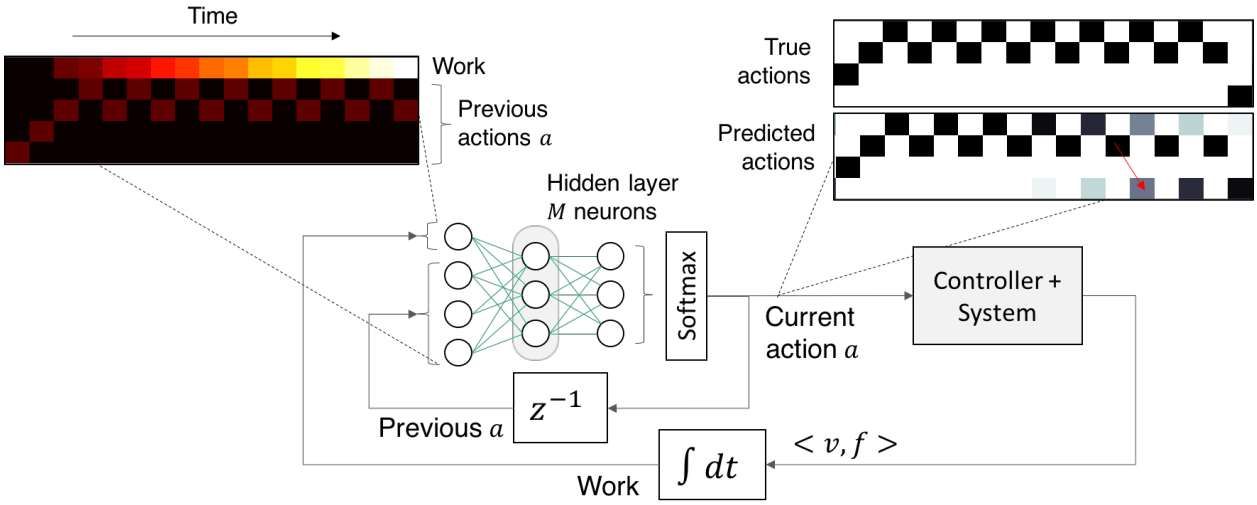


Figure 5.15: The NARX model predicts the next action a_{i+1} given the current action a_i and the mechanical work W_i. Data from an example trial is shown. As the work increases, the probability of transitioning to the completion action increases, until finally it exceeds to probability of continuing the task (denoted by the red arrow).

the same operator performed the task, and was not checking whether the hole was clean. As a result, it would be conjecture to say which of these is the best feature to use for a sufficiently clean hole. However, because mechanical work separates the two datasets and thus represents a notion of cleaning "quality", this feature is selected as the exogenous input for the remainder of the work. In principal however, any one of these features could have been used.

NARX model for action sequence predictions. Having selected work as the exogenous feature, it remains to learn a model of the prediction $p(a_{i+1}|a_i, W_i)$. The network architecture is shown in Figure 5.15. Neural networks are known to perform well for NARX problems, thus the architecture employs a neural network for the nonlinear mapping.

The network input layer dimension is $n_a + 1$, where n_a is the number of actions in $\mathcal{A}$. A single hidden layer consists of M neurons. Because the prediction is a classification problem,

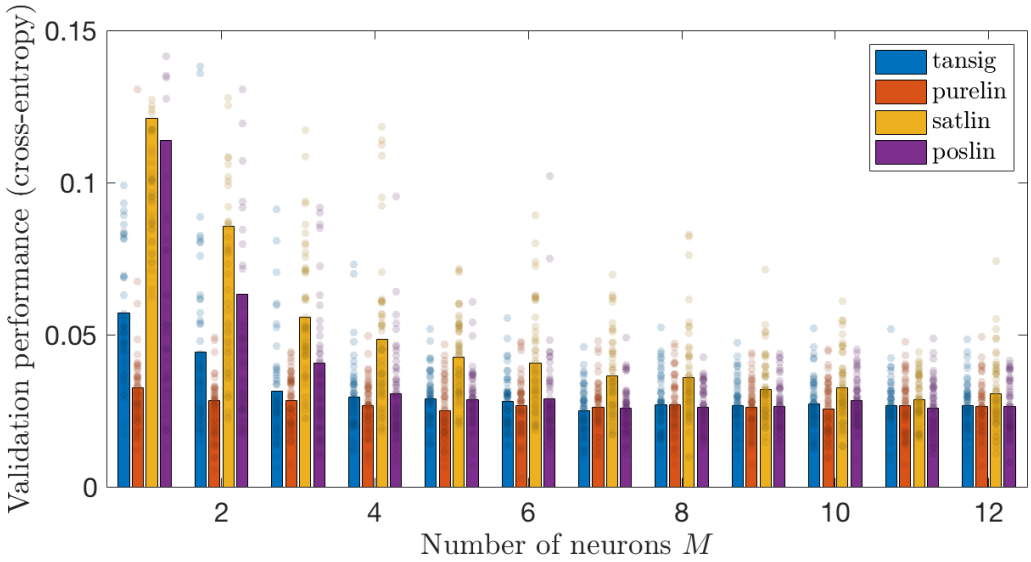
i.e. class $a \in \mathcal{A}$, the neural network structure used includes a softmax classification output. The data is randomly split into 70% for training and 30% for validation. The loss function is cross-entropy, which is the typical loss used for classification problems. Training is performed using the scaled conjugate gradient backpropagation optimizer.

Note that choice of the hidden layer activation function influences the number of neurons needed. Further, number of actions n_a also influences how many neurons are needed. Figure 5.16 shows the validation performance for several activation functions for very different cluster/action models. The activation functions used are: (i) the hyperbolic tangent sigmoidal function *tansig* with an output range of $[-1, 1]$; (ii) the linear function *purelin* with an output range of $(-\infty, \infty)$; (iii) the saturated linear function *satlin* with an output range of $[0, 1]$, and (iv) the rectified linear unit (RELU) *poslin* with an output range of $[0, \infty)$.

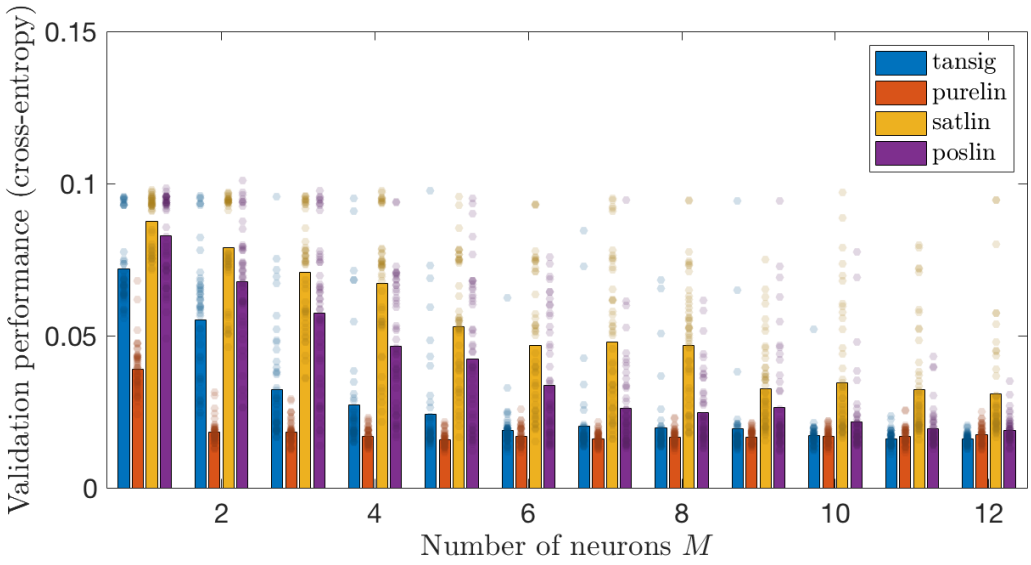
The linear function *purelin* performs the best on the dataset, requiring the fewest number of neurons for convergence, followed by *tansig*. The *satlin* function performs poorly, requiring a relatively high M to converge to best performance. The popular *poslin* (RELU) performs poorly with a few neurons but converges must more quickly than *satlin*. The remainder of the thesis utilizes the *tansig* activation function, although the linear activation would work just as well.

5.3.6 Model training algorithm

An algorithm to learn a set of models $\{\Xi\}$ for an unknown number of states N is presented in Algorithm 2. The algorithm begins by performing the clustering from Section 5.3.3. Next the algorithm finds a ranked set of states as outlined in Section 5.3.4. Since the number of states N is unknown, this set of states $\mathcal{S}$ is recursively pruned until empty. In each recursion, a set of actions $\mathcal{A}$ are found from the states. Since the model likelihood in (5.19) includes the transition sequence model developed in Section 5.3.5, the neural network is trained for each set of actions. This recursion is repeated, increasing the number of models in the set. Note that if the number of clusters K or number of neurons M is also unknown, then this algorithm must be evaluated for each choice of those parameters.



(a) Performance for $K = 8$ clusters and $n_a = 4$ actions.



(b) Performance for $K = 14$ clusters and $n_a = 32$ actions.

Figure 5.16: Validation performance for two different cluster/action models (a) and (b) is shown. The legend indicates different activation functions used.

Algorithm 2 Model training algorithm

Input: $\mathcal{D}$ (data), K (number of clusters), M (neural network structure)

Output: $\{\Xi\}$ (set of models)

function $\{\Xi\}$ = TrainModel$(\mathcal{D}, K, M)$ // Trains for unknown number of states N
  $\{\Xi\} \leftarrow \emptyset$ // Initialize empty model set
  GMM $\leftarrow$ **clusterData**$(\mathcal{D}, K)$ // See (5.20)
  $\boldsymbol{W}_c \leftarrow$ **clusterSequence**$(\mathcal{D}, GMM)$ // See (5.50)
  $\mathcal{S}, r \leftarrow$ **findStates**$(\mathcal{D}, GMM)$ // See (5.47)
  while $\neg$ **empty**$(\mathcal{S})$ **do**
    $\mathcal{A} \leftarrow$ **findActions**$(\boldsymbol{W}_c, \mathcal{S})$ // Breadth-first search. See Algorithm 1
    $\breve{\mathcal{D}} \leftarrow$ **findActionSequence**$(\mathcal{D}, GMM, \mathcal{A}, \mathcal{S})$ // See (5.52)
    $NN \leftarrow$ **learnSequence**$(M, \breve{\mathcal{D}}, \mathcal{A}, \mathcal{S})$ // See Section 5.3.5
    $L \leftarrow$ **likelihood**$(\mathcal{D}, GMM, NN, \mathcal{A}, \mathcal{S})$ // See (5.19)
    $\{\Xi\} \leftarrow$ **append**$(\{GMM, NN, \mathcal{A}, \mathcal{S}\})$ // Append model Ξ to set
    $\mathcal{S}, r \leftarrow$ **pruneWeakestState**$(\mathcal{S}, r)$ // reduces size of $\mathcal{S}, r$ by 1
  end while
  return $\{\Xi\}$ // The model set
end function

5.3.7 Model selection

This section discusses the model selection problem. Several model hyperparameters were introduced, including number of clusters K, number of states $N \leq K$, and number of neurons M in the hidden NARX layer. In some cases, choice of some parameters may be known a priori, however in general this is not the case.

Recall the optimization problem in (5.9),

$$\Xi^* = \arg\min_{\Xi} \; -\ln \mathcal{L}(\mathcal{D}|\Xi) + \lambda\|\Xi\|.$$

Typically, increasing the number of parameters in the model Ξ tends to reduce bias, max-

imizing the model likelihood, i.e. minimizing the negative likelihood. However, above a certain number of parameters, variance begins to dominate the predictive capability of the model, leading to overfitting and poor generalization to new data. The optimal point on this bias-variance tradeoff is the model that has reduced bias but generalizes well to new data.

Popular approaches to reduce overfitting include cross-validation and regularization [**?**, **?**]. Due to lengthy compute times of cross-validation on the task dataset for the model, this work considers a regularization approach. The parameter $\lambda > 0$ introduces regularization into the model selection, i.e. by penalizing a metric of model complexity $\|\Xi\|$. Choice of λ and complexity $\|\cdot\|$ greatly influence the results. The popular Akaike information criterion (AIC) [**?**] and Bayesian information criterion (BIC) [**?**] are information-theoretic approaches to selecting these terms.

In BIC, the optimal model Ξ^* minimizes

$$\min \quad -2\ln\mathcal{L}(\mathcal{D}|\Xi) + \ln(n_s)n_p \tag{5.54}$$

where n_s is the number of samples in the data, and n_p is the number of parameters in the model. The basis for BIC is to find the model that is generating the observed data. Due to this assumption, BIC is known to tend to select biased models that underfit the data [**?**].

In AIC, the optimal model Ξ^* minimizes

$$\min \quad -2\ln\mathcal{L}(\mathcal{D}|\Xi) + 2n_p \tag{5.55}$$

where n_p is the number of parameters in the model. Unlike BIC, AIC does not assume that the model generating the data is in the set, and that higher numbers of parameters will tend to fit the data better. As a result, AIC tends to select more complex models than BIC [**?**].

Performance results for (i) no regularization $\lambda = 0$, (ii) AIC, and (iii) BIC on both the kinesthetic and haptic datasets are shown in Figure 5.17. In these results the number of neurons in the NARX network is held fixed at $M = 20$. The resulting model parameters $\{K, N\}$ selected by the criterion are shown in Table 5.1. In both the kinesthetic and haptic datasets, AIC learns a well-balanced model that is consistent between both datasets in Figures 5.17a

Criterion	Kinesthetic data	Haptic data
Log Likelihood ($\lambda = 0$)	K = 14, N = 3	K = 14, N = 6
AIC	K = 8, N = 3	K = 8, N = 3
BIC	K = 5, S = 2	K = 3, S = 2

Table 5.1: The model selection results using the discussed criterion are shown for both the kinesthetic and haptic datasets. AIC performs consistently between both datasets.

and 5.17b, shown to select models with $S = 3$ states in Table 5.1. No regularization learns varying models of substantial complexity while BIC learns too simple of models that exhibit poor likelihoods, as seen in Figure 5.17a where BIC has overpenalized the model complexity for a biased model. This bias is apparent in the results in Table 5.1, where BIC learns simple models with low numbers of clusters $K = \{5, 3\}$ and $S = 2$ states. For this reason, the AIC criterion is used to the select the model.

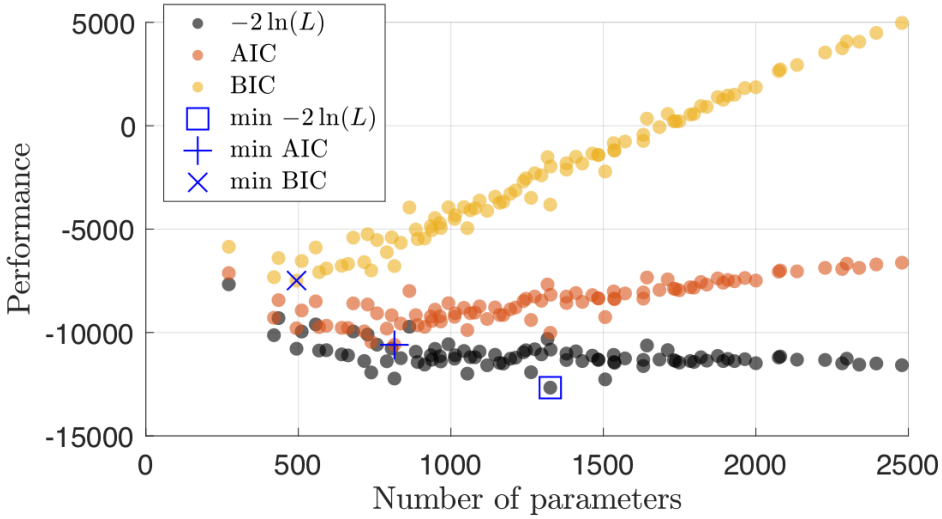
The number of parameters in the model n_p is found via the following

$$n_p = K(d^2 + d + 1) + KA + \hat{M} \tag{5.56}$$

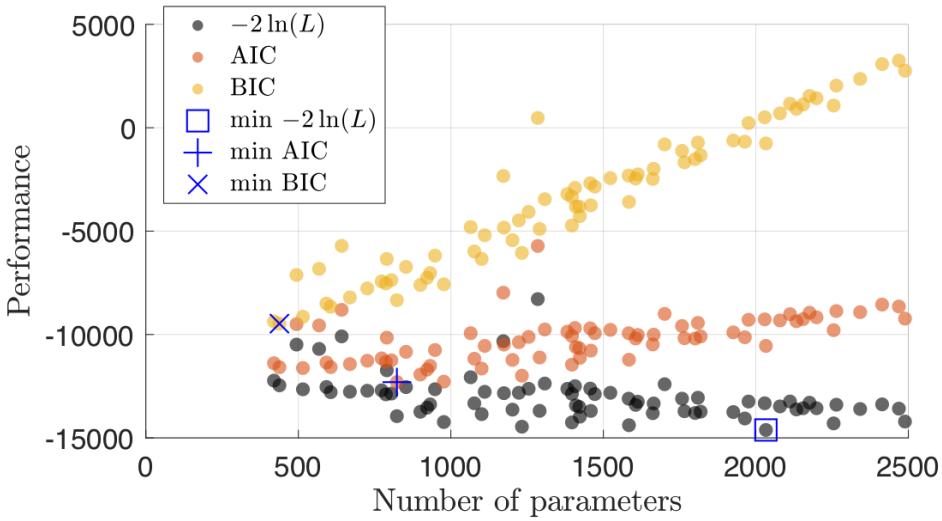
where K is the number of clusters and d is the dimension of the data $\bar{\mathcal{D}}$ in Section 5.3.3, A is the number of actions $\mathcal{A}$ resulting from choice of N in Section 5.3.4, and $\hat{M}$ is the number of parameters in the neural network resulting from choice of neuron layer size M in Section 5.3.5.

5.4 Trajectory optimization for control

While the learned model is able to generate feedforward inputs to a robot controller, a family of trajectories for each action is needed to ensure the robot stays on the intended motion during real-time execution, especially if the robot exhibits uncompensated dynamics. Ideally these trajectories would look similar to the dataset, and match the motion in the GMR



(a) Model performance for the kinesthetic dataset



(b) Model performance for the haptic dataset

Figure 5.17: A comparison of model selection approaches on the kinesthetic and haptic datasets shows that BIC tends to underfit by penalizing the number of parameters too much, while no regularization ($-2\ln(L)$) overfits the data. AIC learns a well-balanced model in both datasets.

acceleration predictions. These trajectories should also be constrained to the transition states $\mathcal{S}$ since these regions define the endpoints of each action.

Formally, this can be posed as the following optimization problem for each $a \in \mathcal{A}$

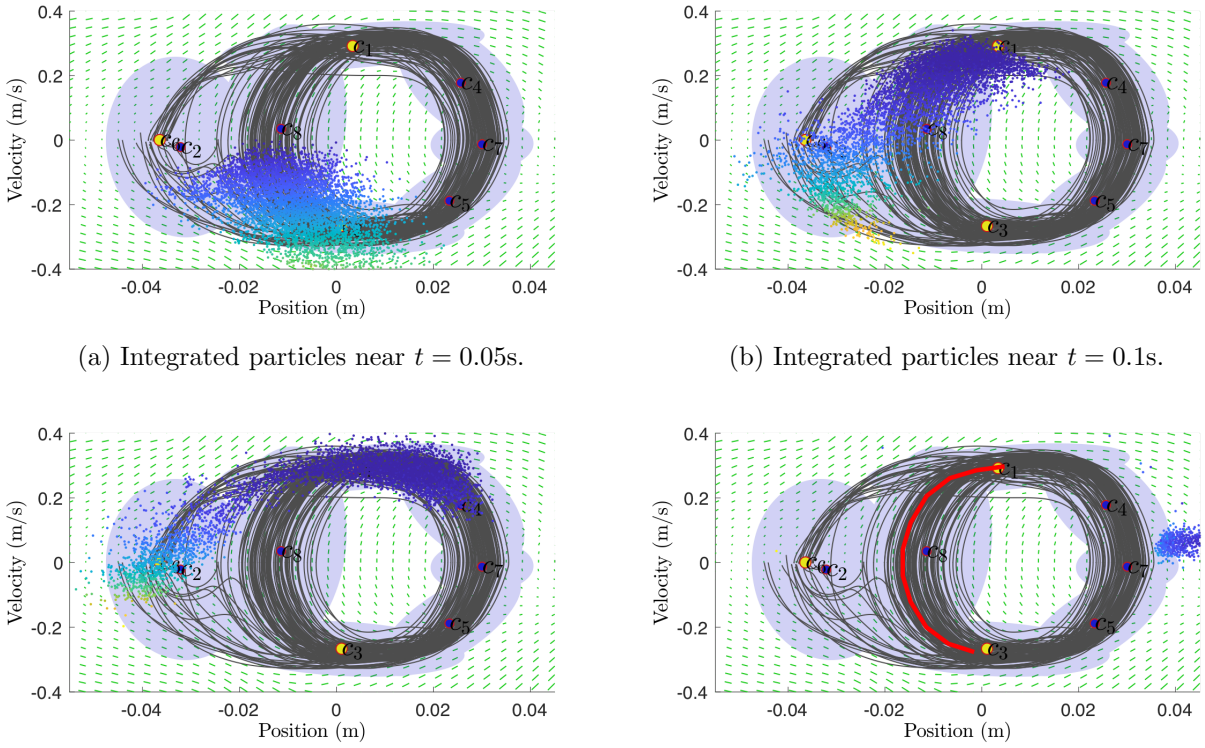
$$\begin{aligned} \{\boldsymbol{\chi}_i, t_i\}_a^* = \underset{\{\chi_i, t_i\}}{\arg\min} \ & \sum_{i=2}^{n} \left\| \boldsymbol{\chi}_i - \int_{t_{i-1}}^{t_i} \boldsymbol{h}(\boldsymbol{\chi_{i-1}}; a)\ dt \right\|_2^2 \\ \text{s.t.:} \quad & \dot{\boldsymbol{\chi}} = \boldsymbol{h}(\boldsymbol{\chi}; a) \\ & \boldsymbol{\chi}(t_0) = \boldsymbol{\mu}_{\chi, \bar{s}(a)} \\ & \boldsymbol{\chi}(t_f) = \boldsymbol{\mu}_{\chi, \underline{s}(a)}, \end{aligned} \tag{5.57}$$

where $\{\boldsymbol{\chi}_i, t_i\}_a$ is the discretized trajectory, $\boldsymbol{h}(\boldsymbol{\chi}; a)$ is the motion model learned in Section 5.3, and $\boldsymbol{\mu}_{\chi,\bar{s}(a)}, \boldsymbol{\mu}_{\chi,\underline{s}(a)}$ are the state means for the starting and ending states $\bar{s}(a)$ and $\underline{s}(a)$, respectively, in action a. The cost function seeks trajectory adherence to the motion model for a, i.e. by minimizing the difference between the system state $\boldsymbol{\chi}_i$ at time t_i, and the prediction from integrating the previous system state $\boldsymbol{\chi}_{i-1}$ over the motion prediction from time t_{i-1} to time t_i. The boundary constraints $\boldsymbol{\chi}(t_0)$ and $\boldsymbol{\chi}(t_f)$ enforce that the trajectory passes directly through the starting and ending states.

Trajectory initialization. To solve (5.57), an initial guess for $\{\boldsymbol{\chi}_i, t_i\}_a$ is needed. Due to complexities in the motion field, the optimization is susceptible to local minima, therefore the initial guess must be close to the desired solution. Fortunately, the generate properties of the GMM can be exploited to sample an initial guess from a Monte Carlo trajectory simulation through the motion prediction.

The Monte Carlo sampling initialization concept is illustrated in Figure 5.18. A set of n_t trajectory starting locations are sampled from the Gaussian initial state $\bar{s}$. The samples are integrated (using 4th order Runge-Kutta) over the motion field $\boldsymbol{h}$, until the mean minimum distance from each particle to the desired state $\underline{s}$ over the entire trajectory has stopped decreasing. This metric ρ for each trajectory is

$$\rho = \min \ \ (\boldsymbol{\chi}(t) - \mu_{\chi,\underline{s}})^\top \Sigma_{\chi,\underline{s}}^{-1} (\boldsymbol{\chi}(t) - \mu_{\chi,\underline{s}}) \tag{5.58}$$



(a) Integrated particles near $t = 0.05$s.

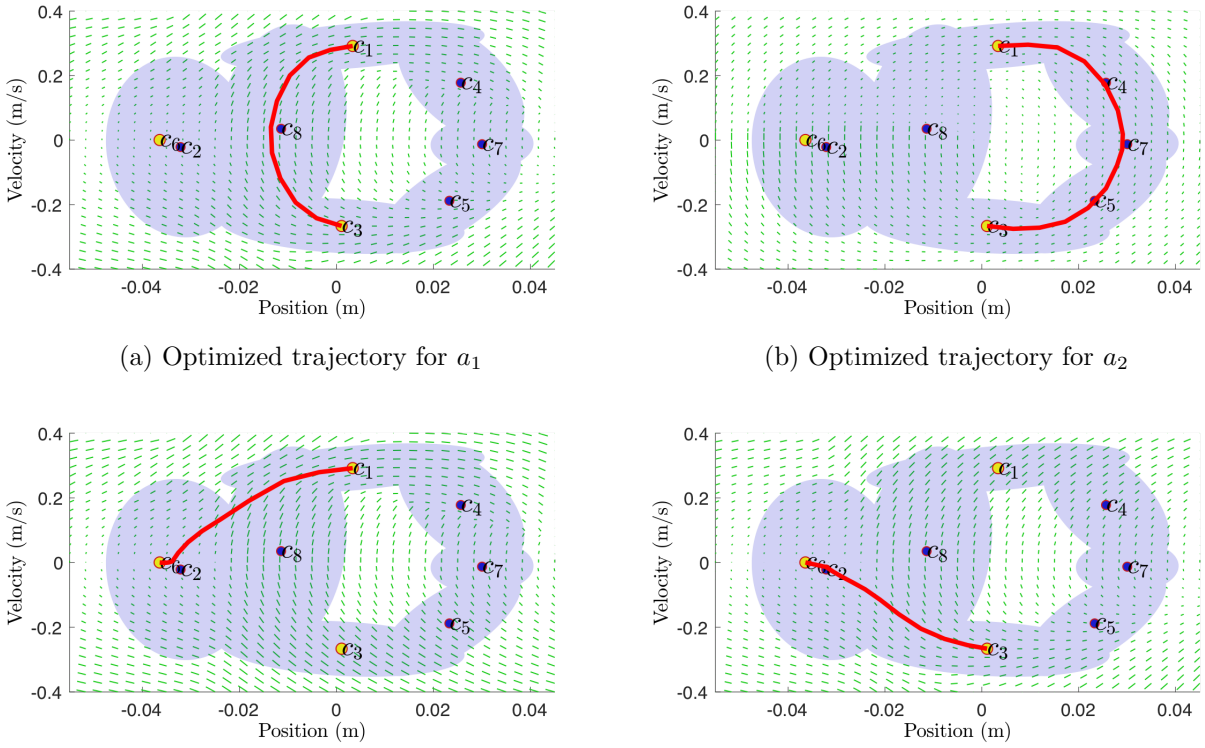
(b) Integrated particles near $t = 0.1$s.

(c) Integrated particles near $t = 0.15$s.

(d) Selected trajectory for initialization.

Figure 5.18: The Monte Carlo sampling approach integrates trajectories through the motion field (a-c), producing a initial trajectory for the optimization (d). Colors indicate metric ρ for each particle, where darker colors denote a smaller ρ.

where $\mu_{\chi,\underline{s}}$ and $\Sigma_{\chi,\underline{s}}$ are the state $\underline{s}$ mean and covariance from the GMM in (5.20). Note that no noise is added during the simulation. The simulation produces trajectories with a cost function $= 0$, however there are violations of the constraints. Selecting a trajectory with the smallest ρ produces a reasonable initialization for the optimization, with a small initial constraint violation. For the simulation, $n_t = 5000$ were used with an integration sample rate of $dt = 1/50$s.



(a) Optimized trajectory for a_1

(b) Optimized trajectory for a_2

(c) Optimized trajectory for a_3

(d) Optimized trajectory for a_4

Figure 5.19: Optimized trajectories are shown for a model consisting of four actions. Each action predicts a motion vector field (shown in green arrows). States are denoted in yellow. The trajectories follow the motion profile through each action, constrained to the starting and ending states.

Optimization. Equation (5.57) is solved with an interior-point algorithm and a numerical gradient computation. The constraints are posed as linear inequality constraints, where the constraint violation is the Mahalanobis distance, defined in (5.51) from the endpoints of the trajectory to the respective starting and ending states, $\bar{s}, \underline{s}$.

For the four actions in the brush cleaning problem, the optimization produces the trajectories shown in Figure 5.19. Enforcing the states as trajectory constraints imposes continuity on $\boldsymbol{\chi}$ when the trajectories are sequenced. Because the states are shared between actions,

the trajectories will also tend to preserve acceleration continuity, i.e. within the final optimization cost.

5.5 Conclusion

This chapter developed a machine learning model of task dynamics from human demonstration data for a cleaning task. Several parameters were introduced to construct the model, and a method for automating this model selection process through the use of regularized information criterion was presented. Finally, a method for generating optimized trajectories from the learned model was developed to facilitate imitation of the task on robot hardware.

There are several limitations with the model that could be addressed by future work.

- The learned GMR parameters are well-behaved only in regions near to the demonstration. Outside of these regions, each action does not necessarily exhibit stable characteristics. One approach to promote better behavior is to enforce stability in the learning. The SEDS technique [**?**] does this with a first-order GMR by enforcing stability constraints on the local linear models during an additional optimization step after the GMM clustering has been performed with EM. This is an area of future investigation, i.e. constraints may be imposed on the structure of the motion profile and the force profile to behave in a stable, more generalized manner. This would also potentially reduce the need for time-based trajectories if the robot is well-compensated.

- The Gaussian mixture modeling approach to clustering the data makes online model updates, i.e. from new data, somewhat difficult. Further investigation or modification of the GMM approach is needed if online updates are to be considered.

- Finally, the sequencing does not adequately capture the notion of a higher-level reward. Re-posing the sequencing problem as a Markov Decision Process (MDP) could open the possibility for more applications and the ability for online adaptation.

Chapter 6
EXPERIMENTS FOR A HOLE CLEANING TASK

6.1 Introduction

This chapter develops an experiment to demonstrate the main contribution applied to a hole cleaning task. The task involves cleaning a sequence of nine holes using a bottle brush mounted to the robot arm. Since the robot is assumed manually placed, the uncertain hole locations must be estimated before the remainder of the task can be automated using the policy learned in Chapter 5. For details on how the methods from Chapters 3 and 4 are integrated into this demonstration, refer to Appendix A. This chapter presents two experiments to complete the experimental contribution **C4**.

- Results of a tracking experiment (**E3**) show the performance of the policies developed in Chapter 5. The experiment shows that use of the force prediction from the task dynamics model as a feedforward input to the robot controller reduces tracking errors as compared with feedback correction alone. This experiment and results are discussed in Section 6.2.

- To demonstrate the main contribution, shared autonomy—enabled by the methods of this thesis—is evaluated in a user study (**E4**) consisting of robotic cleaning for a sequence of nine holes. The contributions are integrated into the experimental platform. The methods, experiments and results are discussed in Section 6.3.

6.2 Policy Tracking Experiment

This section discusses a tracking experiment (**E3**) to evaluate the performance of the policies developed in Chapter 5. For a discussion of feedback gain selection, please refer to

Appendix A.

6.2.1 Cleaning Task Automation

The models and trajectories learned in Chapter 5 are used to automate the cleaning operation. For each action a_k in the sequence of actions starting from $a_{k=0}$ in the autonomous process in (5.16), the associated action trajectory $\{\boldsymbol{\chi}_i, t_i\}^*_{a_k}$ from (5.57) is integrated from the start to finish of the trajectory. When the trajectory has reached its final time t_f, the next action is determined from the autonomous action selection process.

Reference commands. The robot handle position and velocity commands are

$$\boldsymbol{x}^*_h(t_i) = \boldsymbol{\chi}^*_{a,x}(t_i) + (\mathbb{E}[\boldsymbol{x}_t] + \boldsymbol{x}^l_0) \tag{6.1}$$

$$\boldsymbol{v}^*_h(t_i) = \boldsymbol{\chi}^*_{a,v}(t_i) \tag{6.2}$$

where $\boldsymbol{\chi}^*_{a,x}, \boldsymbol{\chi}^*_{a,v}$ are the position and velocity components, respectively, of the system state trajectory for action a, $\mathbb{E}[\boldsymbol{x}_t]$ is the expected value of the target center, i.e. from a localization estimate, $\boldsymbol{x}^l_o$ is the offset of the lth hole from the target center, and t_i is normalized such that $t_i = t_0$ at the start of each trajectory.

Feedforward terms. The feedforward command

$$\begin{aligned} \boldsymbol{u}_{ff} &= \mathbb{E}[\boldsymbol{f}|\boldsymbol{\chi}^*(t_i); a] \\ &= \boldsymbol{g}(\boldsymbol{\chi}^*(t_i); a) \end{aligned} \tag{6.3}$$

applies a force from the model predictive mean $\boldsymbol{g}$ in (5.15), conditioned on the system state $\boldsymbol{\chi}^*$ reference trajectory from (6.1) and (6.2) to perform the prediction. If a reliable estimate of the robotic operational space inertia $\boldsymbol{\Lambda}$ is known, the feedforward can also include acceleration dynamics $\boldsymbol{h}$ from (5.14) in Chapter 5

$$\begin{aligned} \boldsymbol{u}_{ff} &= \mathbb{E}[\boldsymbol{f}|\boldsymbol{\chi}^*(t_i); a] + \boldsymbol{\Lambda}\mathbb{E}[\dot{\boldsymbol{v}}|\boldsymbol{\chi}^*(t_i); a] \\ &= \boldsymbol{g}(\boldsymbol{\chi}_h; a) + \boldsymbol{\Lambda}\boldsymbol{h}(\boldsymbol{\chi}_h; a). \end{aligned} \tag{6.4}$$

Because the robot operational space inertia is not well known for the robot used in these experiments, including acceleration resulted in no tracking improvement, thus only the feedforward command in (6.3) is used.

6.2.2 Evaluation metrics.

Work (control). The policy is designed such that each trial performs the same amount of mechanical work, detailed in Section 5.3.5. This is computed

$$W(t) = \int_0^t \boldsymbol{f}_h(\tau)^\top \boldsymbol{v}_h(\tau)\ d\tau \tag{6.5}$$

where $\boldsymbol{f}_h(\tau)^\top \boldsymbol{v}_h(\tau)$ is the power acting on the workpiece at time τ, from measured forces $\boldsymbol{f}_h$ and velocities $\boldsymbol{v}_h$ at the tool handle. It is assumed that the task starts at $t = 0$.

Mean squared error. The mean squared error (MSE) provides indication to tracking performance by averaging the L2-norm of the error vectors. An MSE $= 0$ indicates perfect tracking. For positions $\boldsymbol{x}$ this is computed

$$\text{MSE}(x) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_i^* - \boldsymbol{x}_i\|_2^2 \tag{6.6}$$

where n is the number of samples collected, $\boldsymbol{x}$ is the measured position feedback, and $\boldsymbol{x}^*$ is the reference position from (6.1). Similarly, MSE for velocity and force can be computed

$$\text{MSE}(v) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{v}_i^* - \boldsymbol{v}_i\|_2^2 \tag{6.7}$$

$$\text{MSE}(f) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbb{E}[\boldsymbol{f}_i] - \boldsymbol{f}_i\|_2^2 \tag{6.8}$$

for reference velocity $\boldsymbol{v}^*$ from (6.2), and reference feedforward force $\mathbb{E}[\boldsymbol{f}]$ from (6.3).

6.2.3 Results and Discussion.

The automated cleaning policy was tested on five holes, with five repetitions of the task for each hole, for a known target location, i.e. $\mathbb{E}[\boldsymbol{x}_t] = \boldsymbol{x}_t$ from (6.1). Four controller strategies were evaluated:

	Work (J)	**Time** (s)	**MSE**(x)	**MSE**(v)	**MSE**(f)
FB-2	6.2±1e-1	9.2±2e-1	1.2e-4±6e-7	2.3e-2±1e-4	57±1
FB-1	6.1±1e-1	9.3±5e-3	1.1e-4±3e-7	2.3e-2±1e-4	58±2
FBFF-2	6.6±2e-1	4.5±2e-1	5.1e-5±2e-6	1.4e-2±2e-4	30±2
FBFF-1	6.5±1e-1	4.5±6e-3	4.3e-5±3e-6	1.0e-2±3e-4	27±1

Table 6.1: Results for the policy tracking experiments are shown for hole 5 (used for training in Chapter 5. The FBFF-1 strategy exhibits the best performance across all metrics.

(**FB-2**) uses the feedback controller ($\boldsymbol{u} = \boldsymbol{u}_{fb}$) from (A.2) with reference positions $\boldsymbol{x}_h$ and velocities $\boldsymbol{v}_h$ from (6.1) and (6.2).

(**FB-1**) is the same as **FB-2**, except that the reference positions in $\hat{y}$ are held fixed to the hole center ($\boldsymbol{x}_t + \boldsymbol{x}_0^l$) in the $\hat{y}$ direction, and reference velocities in $\hat{y}$ are held fixed to 0. Note that the $\hat{x}$ direction corresponds to the principal motion of stroke during cleaning, while physical movement in the $\hat{y}$ direction is constrained due to the hole wall geometry. This controller is labeled "−1" since there is only 1 principal axis of motion from the demonstrated task being controlled.

(**FBFF-2**) uses the feedback and feedforward controller ($\boldsymbol{u} = \boldsymbol{u}_{fb} + \boldsymbol{u}_{ff}$) with the feedback controller from **FB-2** and the feedforward controller from (6.3).

(**FBFF-1**) uses the feedback and feedforward controller ($\boldsymbol{u} = \boldsymbol{u}_{fb} + \boldsymbol{u}_{ff}$) with the feedback controller from **FB-1** and the feedforward controller from (6.3) with forces $\boldsymbol{f}$ in the $\hat{y}$ direction held at 0.

Results of the tracking experiment for the center hole ($l = 5$) are presented in Table 6.1 and summarized in Figure 6.1. Note that mechanical work (6.5) is similar for both controllers because the controller determines when the task is complete based on the mechanical energy

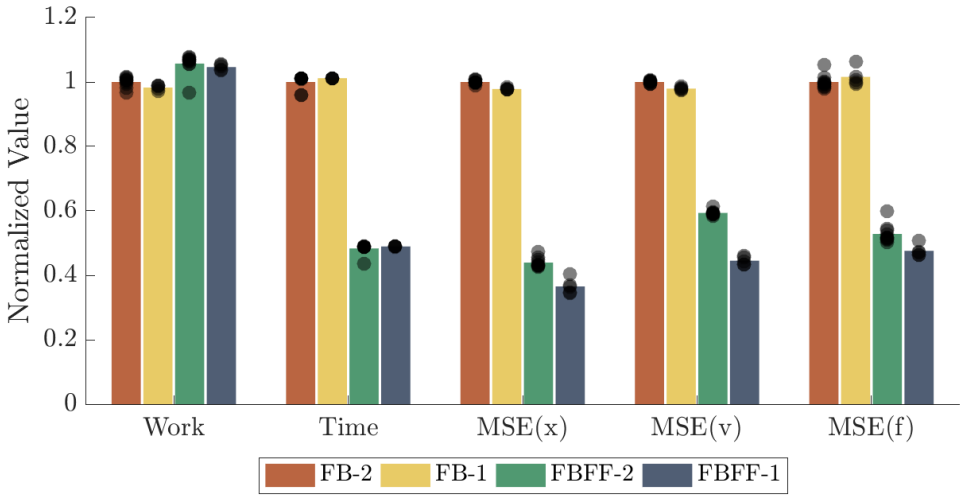


Figure 6.1: A performance comparison of the different control strategies for imitating the cleaning task is shown for hole 5 (the hole used for data collection in Chapter 5). Note the values are normalized such that FB-2 values are 1 to provide a relative comparison.

imparted during the task.

Feedforward kinetics improves performance across all metrics.

Using the predicted forces from the task dynamics model as a feedforward (FBFF-1 and FBFF-2) reduces tracking errors and completion time by 50% as compared with trajectory tracking alone (FB-1 and FB-2). Completion time is reduced by 52%, position MSE is reduced by 63%, velocity MSE is reduced by 55%, and force MSE is reduced by 52%. This illustrates the need for force predictions to improve the ability to imitate expert demonstrations for manufacturing tasks. Note that FBFF-1 exhibited the best performance over all controllers, and was thus selected as the controller for the user study experiments.

Fixing non-principal motions improves tracking when kinetics are included.

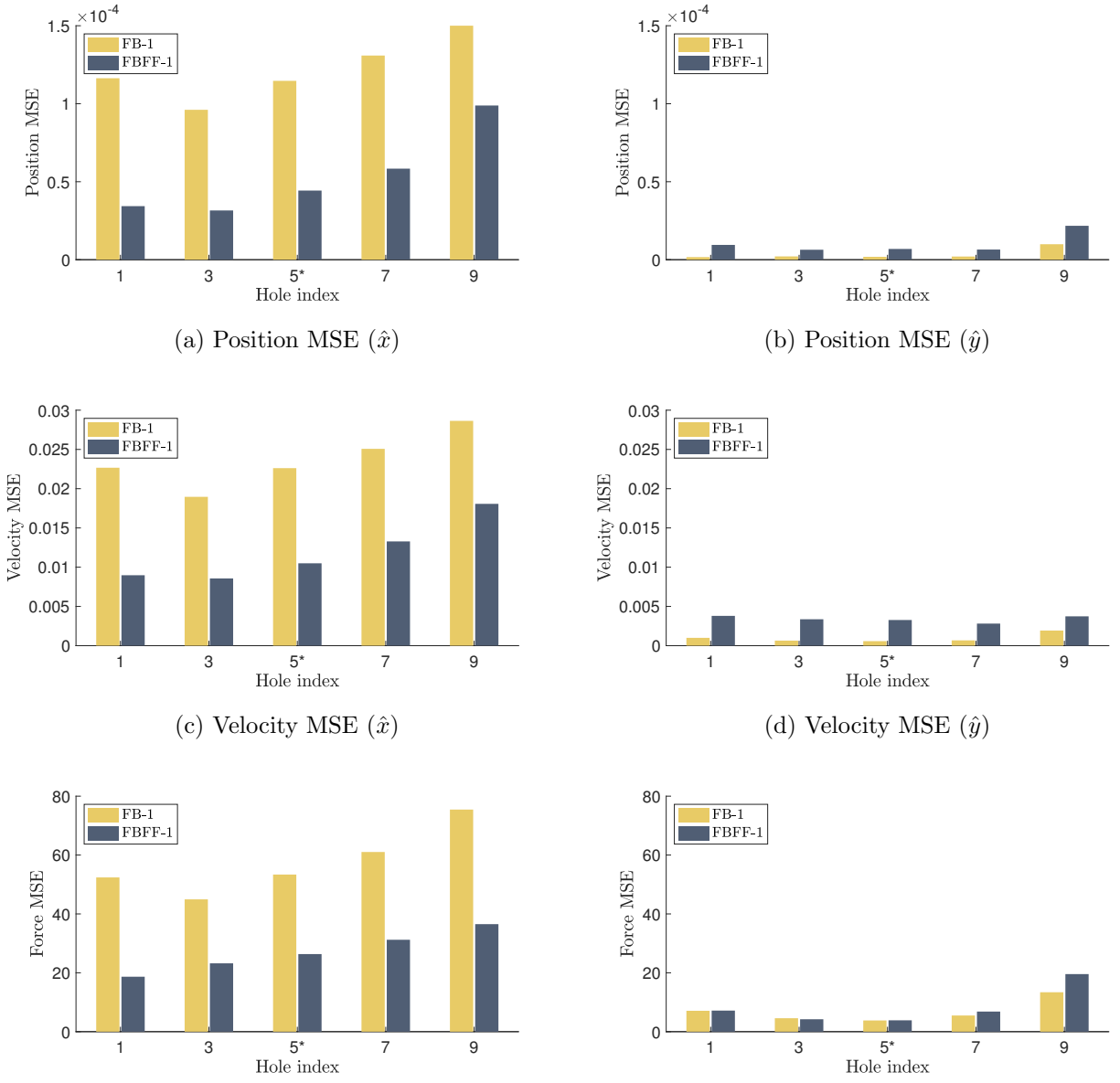
Fixing the motion and forces in the $\hat{y}$ direction (i.e. due to the hole side constraints) to 0 when feedforward was included (FBFF-1) resulted in reduced tracking errors (i.e. 17% in MSE(x), 25% in MSE(v), and 10% in MSE(f)) as compared with feedforward in both $\hat{x}$ and $\hat{y}$ directions (FBFF-2). However, there was no statistically significant change when this modification was made during feedback control alone (FB-1 and FB-2). This suggests that imitating non-zero forces outside of the principal axes of motion has potential to degrade performance. This was visually apparent during the experiments, where increased side-to-side motions during FBFF-2 would cause exaggerated bending in the brush during cleaning. This points to the need for a pre-processing step in the learning in Chapter 5 to reduce a task down to principal axes of motion.

Use of kinetics necessitates consistent robot dynamics.

Tracking error results for FB-1 and FBFF-1 are also examined for several holes, shown in Figure 6.2. Use of feedforward (FBFF-1) reduces the position, velocity and tracking error in the principal axis of motion $\hat{x}$ as compared with alone (FB-1). However, feedforward (FBFF-1) increases tracking errors in $\hat{y}$ as compared with feedback alone (FB-1). In position and force, this degradation in performance is more apparent at holes 7 and 9, which are nearer to the robot reach singularity. This suggests that including kinetics as a feedforward in regions where the system dynamics differs from the demonstration dynamics has potential to increase tracking errors over feedback control alone, especially outside of the principal directions of motion. This illustrates the need for consistency in closed-loop robot dynamics when kinetics is used.

6.3 Human subject experiments

This section discusses a human subject experiment to evaluate the main contribution of the thesis in a practical setting.



(a) Position MSE ($\hat{x}$)

(b) Position MSE ($\hat{y}$)

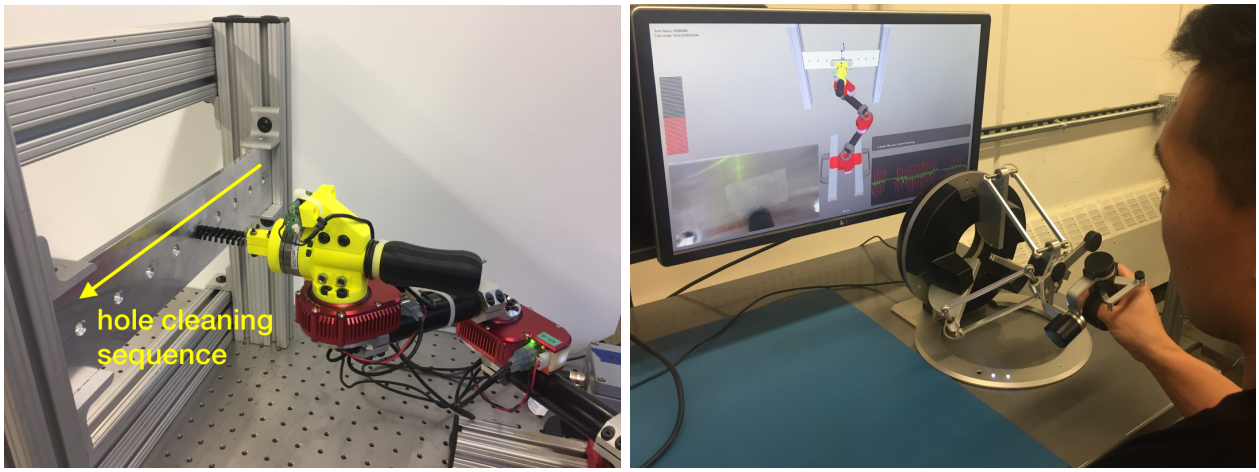
(c) Velocity MSE ($\hat{x}$)

(d) Velocity MSE ($\hat{y}$)

(e) Force MSE ($\hat{x}$)

(f) Force MSE ($\hat{y}$)

Figure 6.2: Mean squared error (MSE) isolated to channels $\hat{x}$ and $\hat{y}$ is shown for the five test holes for (a-b) position, (c-d) velocity, and (e-f) forces. Note that hole 5 was used for data collection in Chapter 5.



(a) Robotic manipulator (b) Teleoperation workstation

Figure 6.3: The experimental setup includes (a) the remote 3-DOF manipulator, and (b) a teleoperator workstation with a haptic interface for teleoperating the robot. The path in blue denotes the trajectory taken by the robot during task automation.

6.3.1 Task and Experimental Setup

The experimental system, shown in Figure 6.3, consists of a custom 3DOF robotic manipulator located at the remote work area, and an operator workstation with a haptic device and visual display from the camera on-board the robot. If the target location $\boldsymbol{x}_t$ is known exactly, the sequence of hole cleaning tasks (for all nine holes shown in Fig. 6.3a) can be automated. On the other hand, if the target location is unknown, the same sequence of hole cleaning can be performed by the human using the teleoperation interface in Fig. 6.3b. In shared autonomy, the human can teleoperate the first few holes while the automation estimates the target location from feedback data. Once the automation is sufficiently confident in this estimate, the remainder of the task can be automated. For a discussion of the target hole localization and off-nominal for this experimental setup, refer to Appendix A.

6.3.2 Experimental Methods

Subjects

Eight (n = 8) subjects agreed to evaluate the proposed shared autonomy for the example hole cleaning task (Figure 6.3). The subjects were predominantly university engineering graduate students, between the ages of twenty and forty. A within-subjects experimental design was chosen to evaluate individual subject performance changes across the independent variables.

User Interface (UI)

Subjects use the haptic device shown in Figure 6.3b to control the robot and the graphical user interface (GUI) shown in Figure 6.4 to monitor the real-time status of the system. A bar graph (Fig. 6.4a) indicates to the user the percent completion for the current hole. When 100% is reached, the bar changes color to indicate the task is complete. A camera feed (Fig. 6.4b) provides a live view from an RGB camera mounted just behind the brush on the remote robot. Messages from the autonomy (Fig. 6.4c) are updated at state changes during the task. A virtual representation of the real-time robot configuration (Fig. 6.4d) is updated via joint position feedback. The current hole for cleaning is displayed as a transparent box (Fig. 6.4e). During automation, current trajectories are displayed as line segments (Fig. 6.4f). Finally, the virtual workpiece with its localization estimate is displayed (Fig. 6.4g). The workpiece transparency is updated via confidence in localization estimate, such that the workpiece is clear for localization confidence near 0 and opaque for confidence near 1 (shown).

During teleoperation, users indicate to the system that they are cleaning a hole via a trigger press on the haptic device in Figure 6.3b. Users are instructed to press the trigger right after they begin cleaning the hole, and release the trigger right after the brush is removed from the hole.

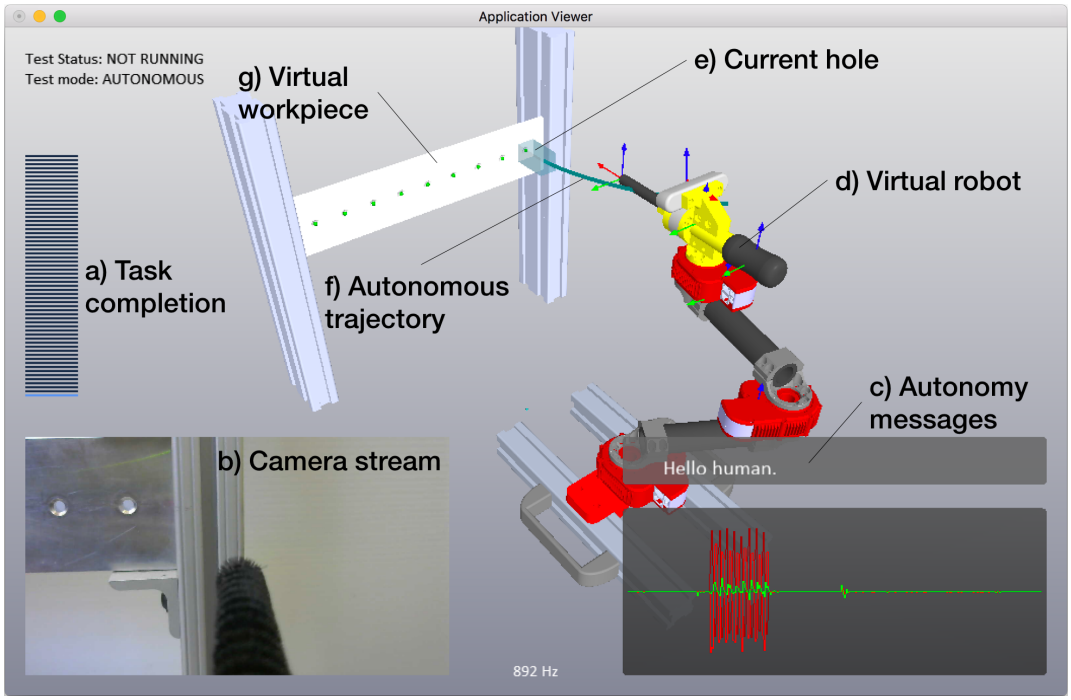


Figure 6.4: The user interface for the experiments includes a) a bar graph to indicate to the user the percent completion for the current hole, b) a live camera stream from the remote robot, c) messages from the autonomy, and d-g) a virtual representation of the robot, task, and workspace.

Independent Variables

The experiment compares the following independent variables: "Full automation" (FA), "Teleoperation" (TO), and "Shared autonomy" (SA). The experimental procedure for these variables is as follows.

Full automation (FA). In automation, the robot automates the entire procedure for a known target location, with no human input. This session is performed before human subjects in order to attain a baseline of automation performance. During automation, the following messages (Fig. 6.4c) are displayed: "*I'm performing a trajectory*" when the robot

is executing a minimum jerk trajectory between holes; and "*I'm performing the hole cleaning procedure*" when the robot is executing the policy for the current hole as in Section 6.2.

Familiarization. In the familiarization pretest step, the user is given several attempts to become familiarized with the system by using teleoperation to clean hole 5 (the hole used for training in Chapter 5). No data is collected during this step. This familiarization is needed to reduce the effects of human learning and adaptation for the task in the results, as users in practice are expected to be familiarized with system operation. However, the user is not allowed to clean any of the other holes during this familiarization, so as to limit exposure to situations outside of the demonstration domain.

Teleoperation (TO). In teleoperation, the human is asked to perform the sequence of nine holes (the same as in full automation) under manual control with the haptic interface. This mode is performed five times for each subject. To ensure the user is performing the correct sequence, the current hole to clean is displayed in the UI (Fig. 6.4e). During teleoperation, the following messages are displayed (Fig. 6.4c): "*It looks like you started cleaning*" when the user presses the button on the haptic device to indicate hole cleaning has started; and "*The hole is clean, time to move on*" when the completion bar reaches 100% (Fig. 6.4a). The virtual workpiece location (Fig. 6.4g) starts in an error condition, and localization is used to actively update its location in the UI.

Shared autonomy (SA). In shared autonomy, the human is asked to perform the same sequence as in teleoperation, however when the autonomy is confident in the localization estimate, it takes over to complete the rest of the task when the current hole has been cleaned (specified via button release and sufficient completion). This mode is performed five times for each subject. During this mode, messages (Fig. 6.4c) from automation and teleoperation are displayed with the addition of the following: "*I'm not sure where the hole is yet, please clean the next one for me*" is displayed when the user depresses the button on

the haptic device to indicate hole cleaning has finished and the hole localization confidence is not sufficient; "*I'm taking over, your work here is done!*" is displayed in the same case the previous except the hole location confidence is sufficient; and "*I encountered an off-nominal condition, please complete this hole for me!*" is displayed when the robot encounters an off-nominal condition and control is returned to the human.

Objective Evaluation Metrics

The following objective metrics are used to compare full automation (FA), teleoperation (TO), and shared autonomy (SA).

Work W (control variable). To ensure that the humans and automation are performing the same task, hole completion percentage is determined by the mechanical work W (6.5) that has been applied during cleaning. During automation, this is internally computed in the policy learned in Chapter 5. For human control, the value was determined by investigating the neural network activation threshold for moving to the final action. The completion energy for each hole W_l is 6.0 Joules, used to determine when the task is marked as complete in the UI (Fig. 6.4a).

Completion time T. Lower completion times are desired to increase process efficiency, as discussed in the objectives in Chapter 1.1. Total completion time T is measured as the time taken from the start of the first movement towards the initial hole to the return to the initial starting condition. This initial position is held fixed for each mode of operation. Completion time of each hole T_l is measured from the start of a hole to the end, as indicated by the human via button press or by the robot via state change.

Operator energy E. The second objective of the research from Chapter 1.1 is to reduce operator workload. In [**?**], a measure of physical workload is actual energy expended by the human. Without physiological sensors, mechanical energy expenditure can be found via

Q1	The robot and I worked fluently together.
Q2	I felt uncomfortable with the robot.
Q3	The robot was trustworthy.
Q4	The robot was intelligent.
Q5	I found what I was doing with the robot confusing.
Q6	Collaboration mode (SA) was mentally easier than manual mode (TO).
Q7	Collaboration mode (SA) was physically easier than manual mode (TO).
Q8	I preferred manual mode (TO) over collaboration mode (SA).
Q9	Collaboration mode (SA) was faster than manual mode (TO).

Table 6.2: Post-experiment questionnaire

applied human forces and velocities at the haptic device, using the definition of mechanical work as in (6.5). The metric indicates the amount of energy that has been applied to produce motion. Operator energy for each hole is denoted as E_l.

Operator force F. Complementary to energy, operator forces provide insight to the magnitude of forces that the user is applying irrespective of motion. This is useful to additionally capture forces that are not being applied toward the mechanical work (control). Root mean squared (RMS) operator forces F are computed at the haptic device

$$F = \left(\frac{1}{n} \sum_{i=1}^{n} \| \boldsymbol{f}_i^o \|_2^2 \right)^{1/2} \tag{6.9}$$

where $\boldsymbol{f}^o$ are the operator forces applied at the haptic device and $i = \{1, \ldots, n\}$ is the number of samples for one test. RMS operator forces for each hole are denoted F_l.

Subjective Evaluation Metrics

In addition to the previous objective metrics, users are asked to answer a short post-experiment questionnaire, using a 7-point Likert scale. The questions are listed in Table 6.2.

Questions Q1-5 are measures of human-robot collaboration fluency from Hoffman [**?**], and are asked for both teleoperation (TO) and shared autonomy (SA). Questions Q6-9 are developed specifically for this experiment, to provide insight to the perceived changes in workload (mental and physical in Q6-7), overall speed (Q9), and subject preference for TO vs SA (Q8). Note that questions Q2, Q5, and Q8 are negative to balance the questionnaire. Users are also asked to rate (on a 7-point Likert scale) the usefulness of eight features in the user interface (UI) for completing the task (not shown).

6.3.3 Objective Results and Discussion

Objective metric results are presented in Table 6.3. Table 6.3a presents the results across trials (i.e. for the sequence of nine holes), and Table 6.3b presents the results across holes. Statistically significant percentage improvements and margin of error are indicated for 99.5% one-sided confidence intervals. Metric results across trials are summarized in Figure 6.5. Metric results for each hole are summarized in Figure 6.6. Mechanical work W is statistically consistent between trials, indicating the same task was performed for full autonomy (FA), teleoperation (TO), and the proposed shared autonomy (SA). For individual subject results, please refer to Appendix B.

Shared autonomy reduces completion time compared with teleoperation.

Use of shared autonomy (SA) reduces completion time T by 54% as compared with teleoperation (TO) across trials (Table 6.3a) and individual holes (Table 6.3b). Use of full autonomy (FA) reduces completion time T by 64% as compared with TO (Table 6.3a and 6.3b). However, since full automation is not possible for the entire task due to target location uncertainty, shared autonomy (SA) is needed. Since even the expert is not as fast as automation, the improvement by automation (FA) represents an upper bound to the performance gains that can be achieved by shared autonomy (SA) for this system configuration. Note that for a higher bandwidth robot, completion time for automation could be reduced further as trajectories between holes could be faster.

	Control	Process	Human operator workload	
	Work W (J)	**Time** T (s)	**Energy** E (J)	**Force** F (N)
TO $(\mu \pm \sigma)$	61.9±4.5	143.0±61.1	22.1±2.5	10.9±1.5
FA $(\mu \pm \sigma)$	58.1±0.5	52.0±0.1	0.5 ±0.0	3.0±0.1
SA $(\mu \pm \sigma)$	60.8±1.8	65.8±12.4	4.3 ±4.4	6.1±1.6
FA *cf.* TO $(\mu \pm ME)$	-	63.6±0.2%	97.7±0.5%	72.3±1.4%
SA *cf.* TO $(\mu \pm ME)$	-	54.0±0.2%	80.5±0.5%	44.0±1.4%

(a) Objective metric results across trials

	Control	Process	Human operator workload	
	Work W (J)	**Time** T (s)	**Energy** E (J)	**Force** F (N)
TO $(\mu \pm \sigma)$	6.79±0.71	12.42±6.75	2.32±0.46	0.15±0.15
FA $(\mu \pm \sigma)$	6.44±0.20	4.49±0.67	0.05±0.01	0.01±0.00
SA $(\mu \pm \sigma)$	6.67±0.41	5.71±2.81	0.45±0.87	0.03±0.06
FA *cf.* TO $(\mu \pm ME)$	-	63.8±3.2%	98.0±8.6%	-
SA *cf.* TO $(\mu \pm ME)$	-	54.0±3.2%	80.6±8.6%	-

(b) Objective metric results across holes

Table 6.3: Objective evaluation metric results (a) across trials and (b) across holes are shown for the independent variables: teleoperation (TO), automation (FA), and the proposed shared autonomy (SA), with mean μ and standard deviations σ shown. The bottom two rows compare the percent reductions in metrics for FA and SA as compared with TO, with margin of errors (ME) shown for a 99.5% confidence interval. Statistically significant improvements of SA as compared with TO are highlighted in green.

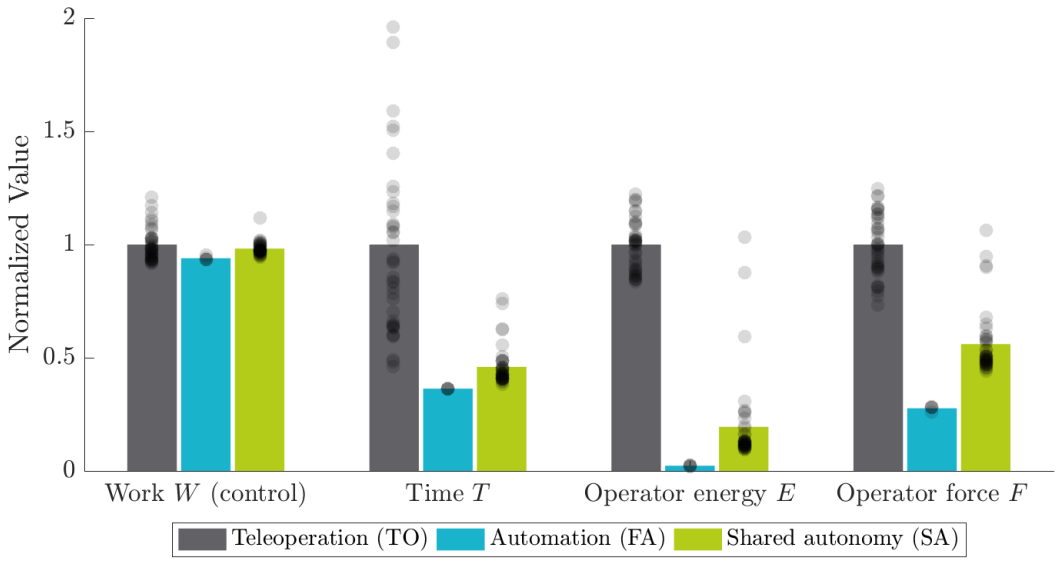
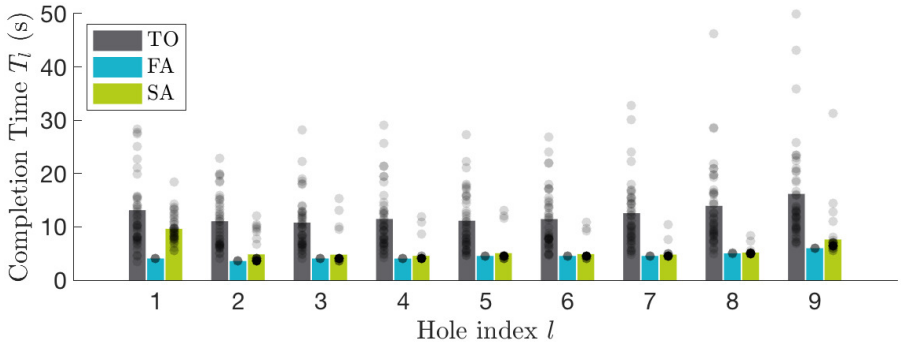


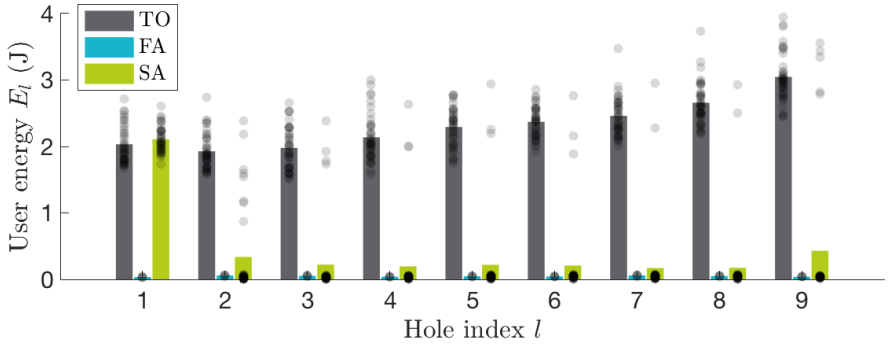
Figure 6.5: A comparison of the independent variables is shown for the objective metrics. Each point represents a single run of the cleaning sequence. Bars indicate the sample mean. Metric values are divided by the value for TO in the corresponding metric to normalize for comparison. This enforces all TO metrics to be 1.

The reduction in completion time is a result of (i) faster automated motion between holes, and (ii) the fact that automation imitates kinesthetic rather than teleoperated demonstrations. Because the kinesthetic demonstrations are more powerful than teleoperated demonstrations in an energy sense, fewer stroke cycles are needed in automation to perform the same amount of mechanical work W as compared with teleoperation. In the case of the expert operator, teleoperation is much more powerful than that of other operators, resulting in less reduction in completion time for this subject.

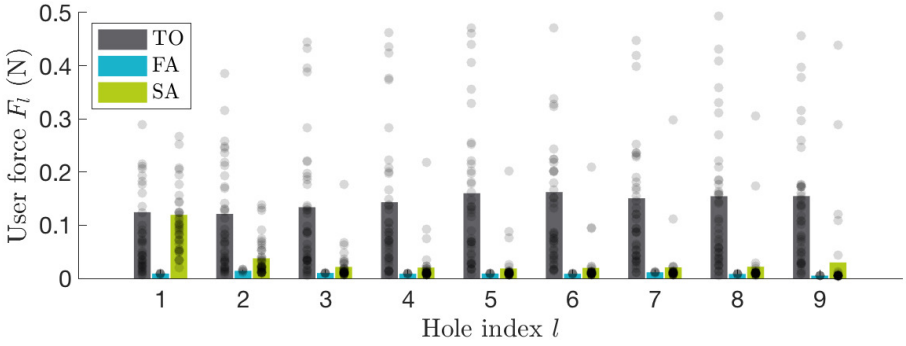
Figure 6.6a shows the individual breakdown of completion times by hole l. Note that for hole $l = 1$, completion time for shared autonomy (SA) is similar to teleoperation. This is because users always manually complete the first hole while the localization converges to an



(a) Completion time for each hole T_l



(b) Operator energy for each hole E_l



(c) Operator RMS force for each hole F_l

Figure 6.6: Objective metrics for each hole (1-9) during the cleaning sequence, comparing teleoperation (TO), automation (FA), and the proposed shared autonomy (SA).

estimate of the target location. The remainder of the holes are completed by the machine at much faster rates. Note that time varies over hole index due to limitations of the robot control, as discussed in Section 6.2.

Shared autonomy (SA) also reduces completion time standard deviations across trials from 61 s to 12 s as compared with TO, making the process more consistent. This reduction in variance is because automation (FA) is more consistent than human teleoperation (TO), causing variance in shared autonomy (SA) to approach FA. However, because portions of the task in SA are still performed manually, variation is still increased over full automation (FA). Note that the distribution of data points for completion time T with SA in Figure 6.6a does not have a Gaussian distribution. The set of higher completion times are cases where operators encountered off-nominal conditions due to a localization error and needed to complete more than one hole for the task.

Shared autonomy reduces operator physical workload compared with teleoperation.

Use of shared autonomy (SA) reduces operator mechanical energy expenditure E by $80.5 \pm 0.5\%$ across trials (Table 6.3a) and $80.6 \pm 8.6\%$ across holes (Table 6.3b) as compared with teleoperation (TO). Similarly, SA also reduces RMS operator forces F by $44.0 \pm 1.4\%$ across trials (Table 6.3a). Reductions in force across holes are not statistically significant.

The reduction of operator energy in SA as compared with TO is due to the introduction of automation into the process. Because the operator only need perform a hole or two manually and the remainder can be automated, the operator expends less energy for the task. This reduction is independent of completion time since the total amount of energy expended by the system (human and machine) is consistent, regardless of how long the process takes.

Figures 6.6b and 6.6c show the breakdown of operator energy expenditure and forces, respectively, for each hole l. As with completion time, energy expenditure E and operator forces F in shared autonomy (SA) are similar to teleoperation (TO) for hole $l = 1$, and decrease for the remaining holes. Typically by the end of hole $l = 1$, shared autonomy (SA) has achieved sufficient localization to automated the remaining holes $l = 2, \ldots, 9$. In

a couple cases, the localization confidence was not quite sufficient, requiring the user to also clean hole $l = 2$ before the robot could take over.

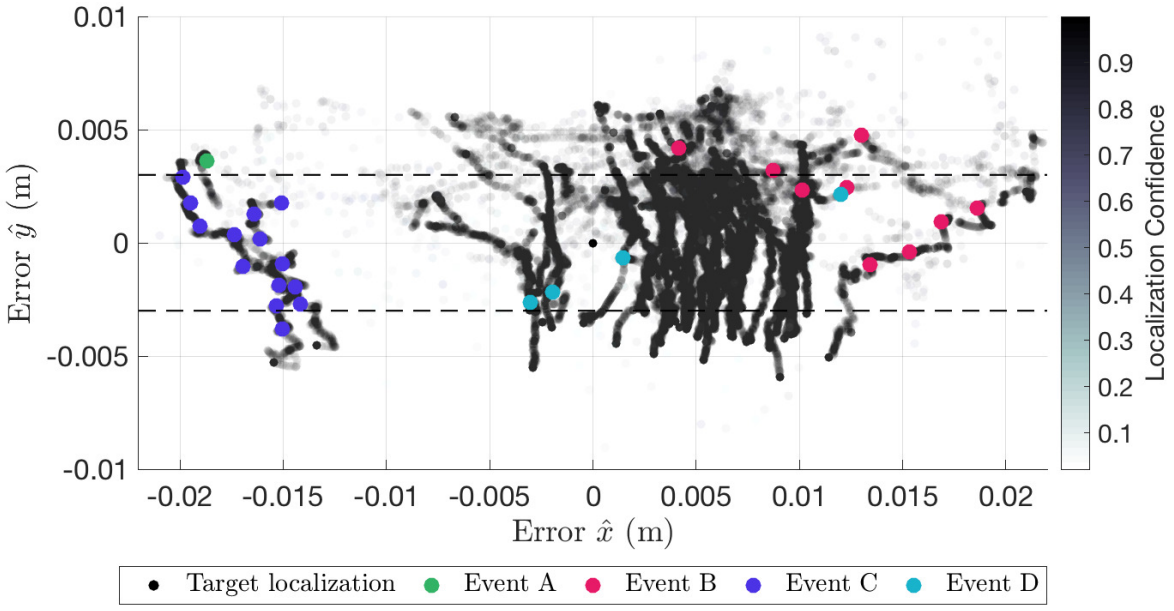
Shared autonomy is robust to localization errors.

Use of off-nominal detection in shared autonomy (SA) makes it robust to localization errors. While the majority of localization errors fall within the theoretical bounds (i.e. from hole geometry), shown as dashed lines in Figure 6.7a, on occasion the localization error is too high for nominal operation. Out of 40 trials, 11 experienced at least one off-nominal event. Figure 6.7 shows localization errors for all trials during shared autonomy. Off-nominal events in shared autonomy (SA) were categorized into the following classes.

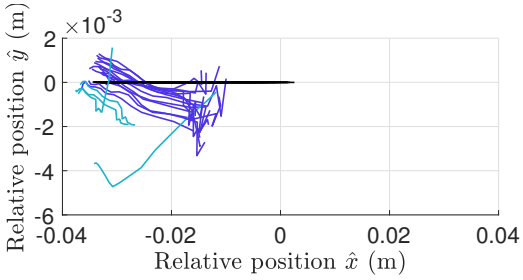
(A) This event occurs during cleaning. In this event, the localization error is too negative in $\hat{x}$ (Figure 6.7a). During the stroke, the brush exits the hole and strikes the coupon (Figure 6.7c). This event occurred once during experiments.

(B) This event occurs during cleaning. In this event, the localization error is too positive in $\hat{x}$ (Figure 6.7a). During the brush stroke, the robot tracking is poorer at positive values in $\hat{x}$ as this is approaching the robot reach singularity. As a result, static friction in the hole overpowers the robot control, and the brush comes to a stop (Figure 6.7c). This event occurred nine times during experiments.

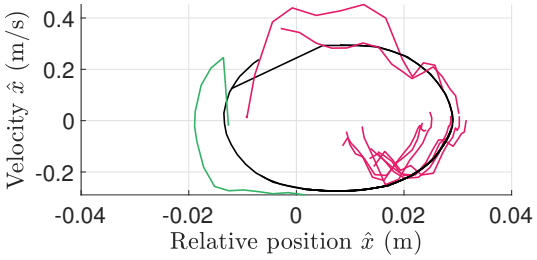
(C) This event occurs during insertion. In this event, the localization error is too negative in $\hat{x}$ (Figure 6.7a). Since the robot pulls to one side, this results in the brush striking either the hole chamfer with too high of feedforward forces, or completely missing the hole and contacting the coupon (Figure 6.7b). This event occurred fourteen times during experiments, although these events occurred during just two trials. Thus, this off-nominal class required operators to manually complete the entire sequence. With better robot tracking, this would not be an issue.



(a) Target estimate errors at detected off-nominal events



(b) Position feedback in $\hat{x}, \hat{y}$ directions



(c) Position-velocity feedback in $\hat{x}$ direction

Figure 6.7: Target localization estimate errors are shown for all trials in shared autonomy (SA). Localization errors at detected off-nominal events are shown in color. In some cases, multiple off-nominal events occur during a single trial. Dashed lines in (a) represent the theoretical error limits in $\hat{y}$, i.e. based on hole and cleaning tool geometry. Solid black lines in (b-c) denote reference trajectories.

(D) This event occurs during insertion. In this event, localization errors are with acceptable bounds (Figure 6.7a). However, the robot still strikes the chamfer with higher forces than nominal (Figure 6.7b). In one case, the robot was not tracking well, and completely missed the hole (Figure 6.7b). This event occurred three times in experiments, two of which were false-positive.

In addition to these events that arose during experiments, during experiment development a bias in the force/torque sensor would also increase off-nominal events. Although this did not occur in experiments, this is a plausible scenario in practice.

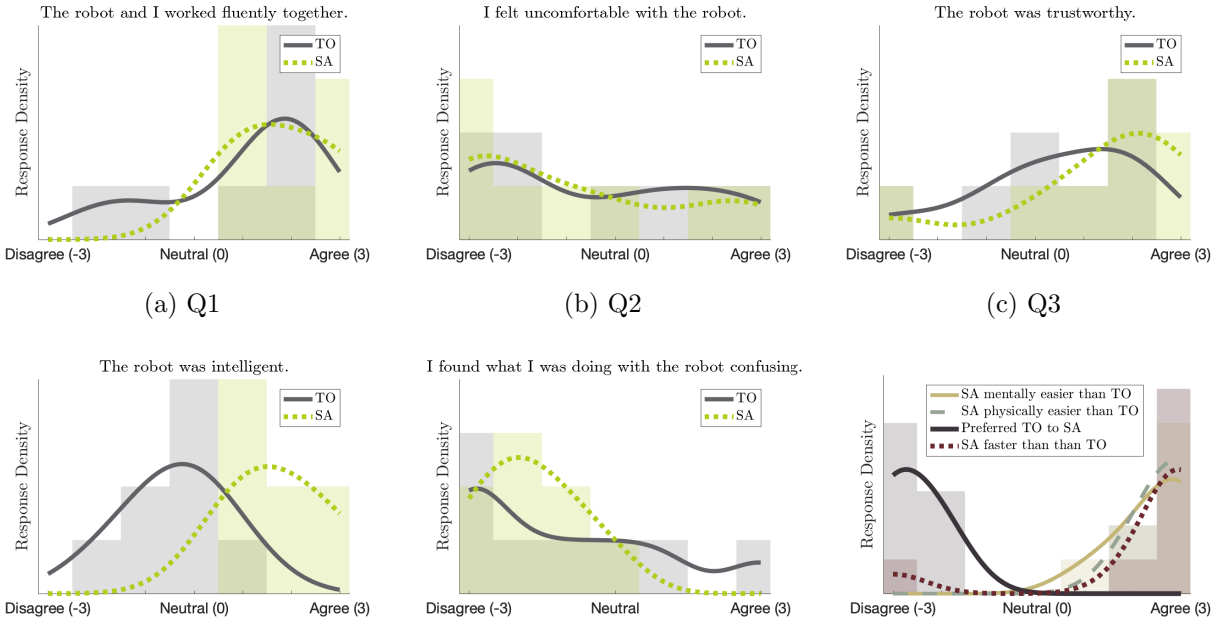
Without off-nominal detection, these events could have been catastrophic for the system. Further, without shared autonomy, this type of event adds—sometimes substantial—time to a manufacturing process due to a system-level restart and the need to perform an entire sequence of operations from the beginning. However, through use of shared autonomy to handle these scenarios, the task can still be manually completed by the operator, without damage to the part or significant increases in completion time.

6.3.4 Subjective Results and Discussion

Response densities for the user questionnaire are presented in Figure 6.8. For the raw responses to the questionnaire, refer to Appendix B. Because the user study involved $n = 8$ users sampled primarily from an engineering population, most of these results cannot be considered statistically significant for a reasonable margin of error and confidence interval. However, the results are still helpful for discussing a subjective evaluation of the experiment.

Shared autonomy increases human-robot fluency.

Most users indicated that they worked fluently with both teleoperation (TO) and shared autonomy (SA), seen in Figure 6.8a (Q1). However, some users disagreed with this statement for teleoperation (TO), but agreed with it for shared autonomy (SA). These users were slower than average at using the manual control. This question from [**?**] evaluates overall fluency



(a) Q1 (b) Q2 (c) Q3

(d) Q4 (e) Q5 (f) Q6-9

Figure 6.8: Results of the user questionnaire Q1-Q6 using a 7-point Likert scale. The histogram data presents the probability, i.e. such that sum of counts for each category = 1. The curves are kernel density smoothing of the histogram data using a kernel bandwidth of 0.7 with a squared exponential kernel, to show the average trends.

of the human-robot interaction.

Users exhibit varied comfort levels for shared autonomy and teleoperation.

Some users felt uncomfortable with both teleoperation (TO) and shared autonomy (SA), seen in Figure 6.8b (Q2), although the majority did not. As in Q1, these users were slower than average at using the manual control. This question from [?] is an adaptation of the *working alliance for human-robot teams*, and evaluates the bond between human and robot.

Shared autonomy decreases distrust in the robot.

Some users indicated they did not trust the robot during teleoperation (TO), however their trust in the robot increased in shared autonomy (SA), seen in Figure 6.8c (Q3). This question from [**?**] evaluates both the user trust in the robot, and the robots perceived character traits.

Shared autonomy increases perceived robot intelligence.

Users indicated the robot exhibited higher intelligence in shared autonomy (SA) as compared with teleoperation (TO), seen in Figure 6.8d (Q4). This question from [**?**] evaluates perception of the robot as a teammate. Not surprising, users disagreed or were neutral to the statement that teleoperation was intelligent.

Shared autonomy decreases user confusion.

Users indicated they found shared autonomy (SA) less confusing than teleoperation (TO), seen in Figure 6.8e (Q5). This is interesting as there is far more more complexity in the shared autonomy than teleoperation, however the additional underlying complexity did not correlate to an increase in subject confusion. It is possible that the user interface provided sufficient transparency to robot actions for this task. This question from [**?**] is a complete composite measure of the *working alliance for human-robot teams.*

Users perceived reduced workload and completion time with shared autonomy.

Users indicated they strongly perceived shared autonomy (SA) to be faster and physically easier than teleoperation (TO) (Figure 6.8f), which correlates with the objective metric improvements in Section 6.3.3. Additionally, users perceived SA to be mentally easier than TO, which corroborates the decrease in user confusion with SA. These questions were developed specifically for this experiment. Using a one-sided binomial test on the "strongly agree" (+3) hypothesis, these results reject the null hypothesis with $> 99\%$ confidence.

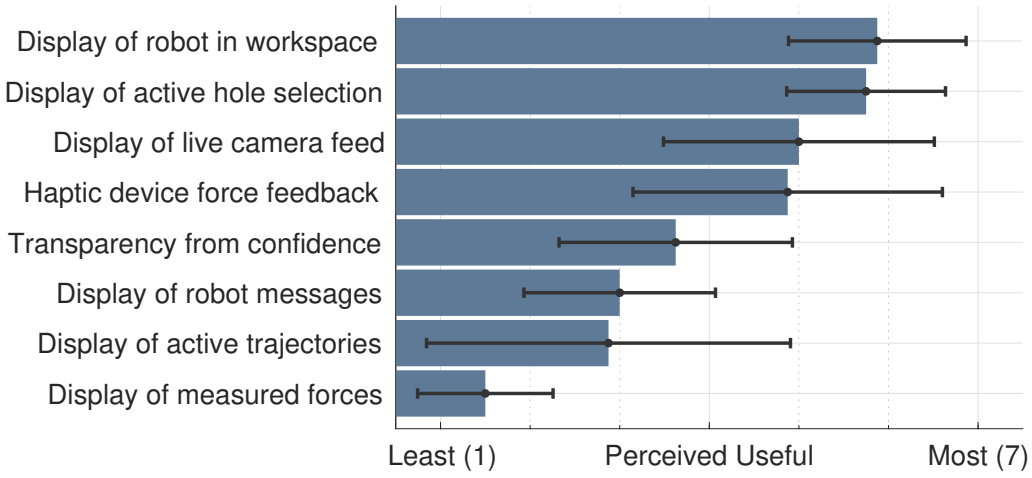


Figure 6.9: Results of the questionnaire for UI feature usefulness.

Users preferred shared autonomy to teleoperation.

Users indicated strong preference in shared autonomy (SA) over teleoepration (TO) (Figure 6.8f). This question was developed specifically for this experiment. Using a one-sided binomial test on the "strongly disagree" (-3) hypothesis, this result rejects the null hypothesis with $> 99\%$ confidence. Note that this question is negative to promote sentiment balance in the questionnaire and ensure user attention.

Users consistently preferred certain features in the UI for the task.

In addition to the previous questions Q1-9, users were asked to rate the usefulness of UI features for performing the task. These results are presented in Figure 6.9. Overall, users found the virtual robot and active hole selection to be the most useful for the task, and display of measured forces to be the least useful. Users were varied regarding the live camera feed and force reflection through the haptic device, typically preferring one over the other.

6.4 Conclusion

This chapter presented two experiments to evaluate the contribution of the thesis. The first experiment found that when performing the policy learned in Chapter 5 on the robot platform, including predicted task dynamics forces as a feedforward control reduced tracking errors by 50% as compared with trajectory feedback control without the feedforward. The second experiment with human subjects for cleaning a sequence of holes found that use of shared autonomy reduced completion time by 54%, reduced operator energy expenditure by 80%, and reduced RMS operator forces by 44% as compared with teleoperation. Additionally, subjects indicated perception of these performance improvements and exhbiited a strong preference for shared autonomy over teleoperation.

There are several limitations of the experiments that could be addressed by future work.

- Including model-predictive forces from the task dynamics model in the policies from Section 6.2 was found to necessitate better robot control. Due to robot control limitations, the results were better near middle holes since robot control was degraded at hole locations on the far ends (closer to robot reach singularities). Further, acceleration feedforward from the task dynamics model was not included in the evaluation since a good estimate of the generalized manipulator mass matrix was not found, and dominant dynamics for this robot are physics in the actuator (friction, motor inertia, backlash, and drive train elasticity). This illustrates the need for hardware improvements and better robot dynamics compensation in order to realize the full potential of the method.

- The methods were evaluated for a hole-cleaning task. In future work, other tasks should be considered to ensure the methods meet the challenges for a broader class of manufacturing tasks.

- The sequence of hole cleans had to be specified prior to the user study. In application, efficiency could be increased by providing the autonomy with a means to recognize the

user intention and automatically generate a sequence based on such intentions.

- The user studies did not span the class of users that would be provided with this system in practice. To ensure that the methods are useful for a true manufacturing business case, the system needs to be evaluated by the end user, in this case aerospace mechanics and technicians. Such workers are known to quickly discard any tools that they perceive to degrade their ability to perform the work, even if an obvious ergonomic benefit exists through use of the tool. This presents a practical design challenge for adopting shared autonomy systems.

Chapter 7

CONCLUSIONS

7.1 Summary

The objective of this thesis was to reduce operator workload and task completion time for manufacturing tasks through the use of telerobotics with shared autonomy. The main contribution towards this objective was to address several key challenges through the use of an imitation learning approach that modeled both the task motions (i.e. position, velocity, and acceleration) as well as the interaction forces.

7.1.1 Confidence-based control trading

In Chapter 3, the concept of mixed-initiative traded control based on localization confidence was introduced, allowing a robot to request control when the confidence exceeded a specified threshold. The chapter presented a method for particle filter localization, in this case using local camera feedback to correct the estimate. However, due to computer vision detection errors stemming from sensitivity to lighting and false positives, a method for resolving disagreements due to localization errors was developed. This work was presented at the IEEE Conference on Advanced Intelligent Mechatronics (AIM 2017) [**?**]. In the face of perception issues, the proposed *consensus-based traded control* was shown to enable traded control to reduce task completion time by 50% as compared with teleoperation and increase task success rate from 3% to 97% as compared with automation. However, due to degraded computer vision reliability for the example manufacturing environment, a major performance limitation of this effort was the sole reliance on computer vision for localization. Further, overconfidence in the visual estimate required the human to remain vigilant in order to quickly take control before some events could cause damage to the system.

7.1.2 Addressing off-nominal situations

The limitations with the methods in Chapter 3 prompted an investigation into force-based methods to uncover off-nominal situations, e.g. due to unexpected obstacles or changes in the hole cleaning sequence. In Chapter 4, a method was proposed to use the interaction forces between a human operator and autonomy in order to detect such off-nominal situations. This work was presented at the IEEE Conference on Intelligent Robotics and Systems (IROS 2018) [**?**]. Due to the addition of interaction forces, this chapter was able to exploit haptic shared control in order to provide compliant active assistance during teleoperation, as opposed to the traded control paradigm from Chapter 3. Additionally, the chapter used imitation learning to develop the models needed to predict interaction forces and the policy that provided compliant assistance. The contribution was experimentally shown for a user study (n=11) to reduce task completion time by 17% and operator forces by 68% as compared to shared teleoperation without the method. However, a key limitation with this effort was that the imitation learning technique used to produce the policy for assistance suffered from prediction bias due to averaging over complex interaction and robot dynamics, making it incapable of fully autonomous operation.

7.1.3 Learning policies for task automation

Because the human remained actively engaged in haptic shared control, the naive learning approach from Chapter 4 was sufficient to provide a policy for assistance, but was incapable of automating tasks with non-negligible interaction forces, i.e. due to contact with the environment. Chapter 5 proposed a learning approach to develop a policy for fully automating the manufacturing task. An existing technique known as Gaussian mixture regression (GMR) was modified to predict second order dynamical system (DS) motions as well as the expected interaction forces during task automation. The policy was found by optimizing a sequence of trajectories over the dynamical system. Due to the predictive capability of the dynamical system model, interaction force predictions from the model were used to provide

an additional feedforward force in order to assist the robot controllers in compensating for task dynamics. In addition to the policy, this dynamical system learning approach also produced the models that were needed to (i) predict interaction forces for detecting off-nominal situations in Chapter 4, and (ii) predict interaction forces and motions for particle filter localization updates as opposed to the computer vision-based approach discussed in Chapter 3. Note that the specifics of integrating the task dynamics model with the previous Chapters 3 and 4 are discussed in Appendix A.

7.1.4 Experimental demonstration of the collaborative system

To evaluate the effectiveness of the task dynamics model in realizing the research objective, experiments on a real robotic system were used to (i) determine the reduction in tracking errors by including force prediction in the control policy for the automated task, and (ii) evaluate the completion time and operator workload during a user study (n=8) where shared autonomy was used to perform a sequence of cleaning nine holes. In the first experiment, including force predictions from the task dynamics model was shown to reduce tracking errors in position (61%), velocity (57%), and force (53%), and reduce completion time (50%) as compared with automation without the force predictions. In the second experiment, the task dynamics models enabled localization, off-nominal event detection, and task automation to be integrated in the traded control architecture discussed in Chapter 3. The enabled shared autonomy was shown to reduce completion time by 54.0% and operator workload by 80.5% as compared with teleoperation. Results from a user questionnaire indicated that users perceived these changes in performance. Users also indicated that they perceived a reduction in mental workload with shared autonomy, and preferred shared autonomy to teleoperation.

7.2 Generalization to new tasks

The task dynamics approach presented in this thesis was developed for data from a hole cleaning task. However, this task contains elements that are common across other manufacturing tasks.

First, the target (hole) position is assumed static, which is the case for many manufacturing processes, including typical drilling, fastening, deburring, and sealant application tasks, to name a few. For tasks such as sanding or polishing, a proxy target may represent the desired region of material removal, in which case the target is not static. For the static assumption to be relaxed, the target motion dynamics in the localization would need to be changed. The data collection step of the GMM also assumes the hole location is static during demonstrations, in which case the model structure as a whole may need to be reconsidered. A good starting point for adapting the technique to this case would be to adapt the graphical task model in Chapter 5.

Second, the input feature to indicate task completion was mechanical work. This is likely to not be the objective across all tasks, although it was a useful feature to examine for hole cleaning. In general, different tasks will require different metrics. For example, fastening may require the breakaway torque to be a specified value, in which case the completion feature could be a binary state indicating whether the objective has been achieved. Similarly, for deburring, the completion condition could be number of completed passes.

7.3 Future Work

7.3.1 Compliant actions

Due to the custom robotic manipulator, system disturbances were non-negligible, requiring the use of trajectory feedback control to keep the robot on the intended motion. A better controlled robot would potentially support the use of compliant actions, i.e. using the learned dynamical system directly to provide the motion. In this case, stability of the model representations would need to be considered, e.g. using Lyapunov techniques [**?**] or contraction analysis [**?**]. These compliant actions are important for human interaction since they are more flexible to variable initial conditions than time-based trajectories.

7.3.2 Separating policies from physics

The proposed model of task dynamics lumps the demonstrated control policy and task physics into a single model representation. While this is concise and solves imitation learning for the task presented in this thesis, separating the control policy from the physics offers exploration of each model independent of the other. The value of this is the ability to learn actions independent of physics manifolds, thus supporting the generalizing of new policies to an existing physics model, or application of an existing policy to a new physics model. To do this, data collection must include independent measures of the forces from the task and the forces exerted by the demonstrator.

7.3.3 Autonomous action sequencing as a decision process

The previous decomposition has an important ramification: new actions can be proposed on an existing model of task physics. This is needed to support techniques that leverage Markov Decision Process (MDP) representations, e.g. enabling inverse reinforcement learning (IRL) for learning a hybrid policy to sequence the discrete set of continuous actions, or reinforcement learning (RL) for improving the automation performance. Reposing the sequencing in this way has potential to close the gap between imitation learning and reinforcement learning. The key to finding objective functions from demonstrations would be to adequately capture reward functions from data.

7.3.4 Intention recognition

The experiments required an explicit button press from the operator to indicate when the task is performed. In general, this can be problematic if the user accidentally presses the button, which occurred on occasion in the experiments. Decomposition of the policy from the task physics would support the ability to estimate intention rather than require explicit human communication. The challenge is that different users will exhibit different control policies, therefore how to generalize the representation across a broad set of users is needed.

7.3.5 Exploration on different tasks

The experiments in this work explored a single example task. Future work should focus early-on in the development of a dataset from many different humans (preferably mechanics) performing various manufacturing tasks in controlled and representative environments. Collected data should include at a minimum: operator interaction forces (for policy isolation), task interaction forces, positions, velocities, accelerations, time, and visual feedback (for improved reward functions).

7.3.6 Robust hierarchical clustering

The state-action decomposition in Chapter 5 can be seen as a form of agglomerative clustering, where keyframes (i.e. states) in the acceleration profile are used to group the existing GMM clusters into a hierarchy that is one layer deep. For the problem set examined, the same keyframes in the acceleration profile also separate the force data into well-behaved distributions, however this is not generally the case. Robust hierarchical clustering [**?**] on the set of linear state space models (both the dynamical and observation equations) is a promising direction for constructing general hierarchies and conceptually supports a task hierarchy of arbitrary depth. A key challenge to address is the selection of a reliable measure of local model similarity.

7.3.7 Regions of safety

Off-nominal situations are detected via statistical inference, however there are no strict guarantees on safety. Since the training data contains nominal demonstrations, the learned distributions could be used to provide a region of acceptable states (i.e. using confidence intervals). Model variance could also be used to provide bounds on the model uncertainty, and trajectory plans and control gains may be optimized (e.g. using robust control techniques) to ensure reduced-risk controllers.

7.3.8 Generalization with inverse reinforcement learning

This thesis used behavior cloning to create a policy for task automation, however this technique is known to generalize poorly to new situations, e.g. contact with the coupon (which was not included in the training data). Inverse reinforcement learning (IRL) generalizes better to new domains, however there are some challenges to consider when applying IRL to the type of task examined in the thesis. First, the domain is continuous and potentially high dimensional, i.e. $\mathbb{R}^{(3+k)n}$ where position, velocity, acceleration, and k effort sources are considered for n degrees of freedom. Second, the feature space that comprises the reward function may be difficult to evaluate directly, e.g. it is challenging to detect hole cleanliness. Finally, discontinuities in the task (e.g. striking the coupon) introduce state transition uncertainty. An IRL exploration would want to include such anomalies and subsequent human recovery in the training data sets in an effort to learn higher-reliability controllers.

Appendix A
EXPERIMENTAL SETUP

A.1 Feedback gain selection

The robot is controlled using operational space control [**?**]. The tool orientation is held fixed, i.e. with a stiffness of 10 N/rad, so that the brush is coaxial with the hole centerlines. The operational space forces applied to the tool handle position $\boldsymbol{x}_h$ are

$$\boldsymbol{u} = \boldsymbol{u}_{ff} + \boldsymbol{u}_{fb}, \tag{A.1}$$

where $\boldsymbol{u}_{ff}$ is the feedforward control input and $\boldsymbol{u}_{fb}$ is the feedback control input. The feedback $\boldsymbol{u}_{fb}$ is specified as a state-feedback controller

$$\boldsymbol{u}_{fb} = K_p(\boldsymbol{x}_h^* - \boldsymbol{x}_h) + K_d(\dot{\boldsymbol{x}}_h^* - \dot{\boldsymbol{x}}_h) \tag{A.2}$$

where $\boldsymbol{x}_h^*, \boldsymbol{v}_h^*$ are reference handle positions and velocities, and $\boldsymbol{x}_h, \boldsymbol{v}_h$ are measured handle positions and velocities. The feedback gains K_p, K_d are scalar values. The gains are selected by examining the position and velocity errors in feedback for a minimum jerk trajectory profile shown in Figure A.1. Note that there are no external disturbances acting on the robot during execution of this trajectory. The trajectories are designed such that identical commanded motions occur in free space near the hole locations. This is to ensure the robot behavior is understood in regions where the policy is to performed.

Position tracking errors for several choices of K_p and K_d are presented in Figure A.2. Error values are different between the holes because the robot is in a new kinematic configuration for each motion. However, the configuration is consistent between trials because the robot is not redundant. Note the demonstrations in Chapter 5 occur on hole 5. Increasing K_p reduces the position and velocity tracking error, as expected. Increasing K_d reduces

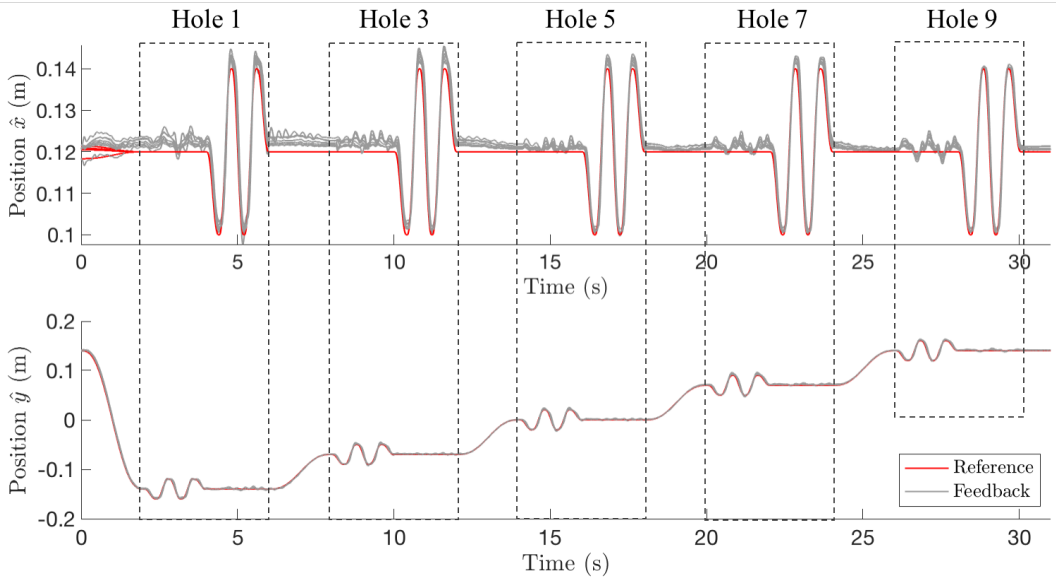


Figure A.1: Several feedback gain values are evaluated for a trajectory tracking experiment. The minimum jerk trajectory includes "plus"-shaped motions near holes 1, 3, 5, 7, and 9. Note the motion profiles occur in free space *near* the specified hole locations.

the velocity tracking error, and thus aids with the position tracking error during motion. Increasing $K_p > 1250$ or $K_d > 20$ begin to introduce instabilities into the control loop near holes 1 and 9, due to sampling rate, noise on the velocity feedback, and actuator harmonics. Based on these results, values of $K_p = 1000$ and $K_d = 10$ were selected to achieve the needed tolerance for successful brush insertion.

A.2 Target Hole Localization

The target location $\boldsymbol{x}_t$ is uncertain since the robot is assumed to be manually placed. To compute an estimate of the target location $\mathbb{E}[\boldsymbol{x}_t]$, localization is performed using the same particle filter approach described in Chapter 3. However, since computer vision is not used in the final experiment, the measurement vector $\boldsymbol{z}_t$ in (3.2) and likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ in (3.7) differ as follows.

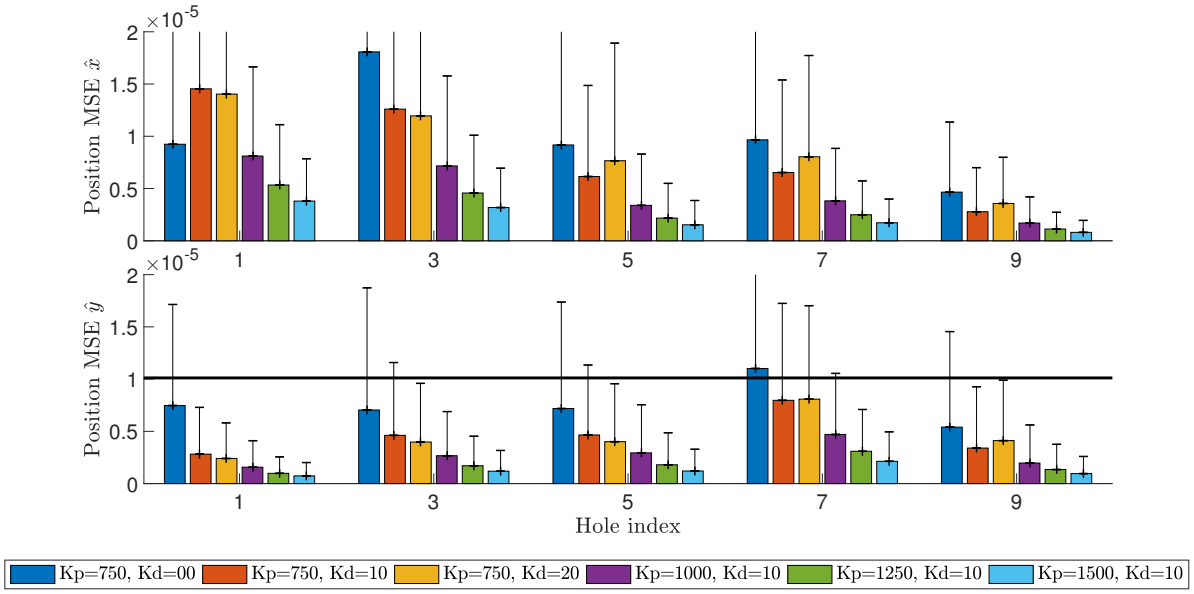


Figure A.2: Position tracking errors (MSE) are shown for 5 identical trajectory profiles near the specified target hole indices. The horizontal bar in $\hat{y}$ denotes the required tolerance for successful brush insertion. Bounds for each bar represent 1 standard deviation from the mean. The gains selected for control are $K_p = 1000$ and $K_d = 10$ (green).

Measurement vector. The measurement $\boldsymbol{z}_t = (\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \dot{\tilde{\boldsymbol{v}}}_h, \tilde{\boldsymbol{f}})^\top$ includes lowpass filtered robot handle position $\tilde{\boldsymbol{x}}_h \in \mathbb{R}^2$ (w.r.t the robot origin), filtered robot handle velocity $\tilde{\boldsymbol{v}}_h \in \mathbb{R}^2$, robot handle acceleration $\dot{\tilde{\boldsymbol{v}}}_h \in \mathbb{R}^2$ (computed during the velocity filtering), and filtered measured forces $\tilde{\boldsymbol{f}} \in \mathbb{R}^2$ at the robot handle. The filtered features are computed via the following

$$\tau_c \dot{\tilde{\boldsymbol{x}}}_h + \tilde{\boldsymbol{x}}_h = \boldsymbol{x}_h \tag{A.3}$$

$$\tau_c \dot{\tilde{\boldsymbol{v}}}_h + \tilde{\boldsymbol{v}}_h = \boldsymbol{v}_h \tag{A.4}$$

$$\tau_c \dot{\tilde{\boldsymbol{f}}} + \tilde{\boldsymbol{f}} = \boldsymbol{f} \tag{A.5}$$

where $\tilde{\cdot}$ denote filtered values, and $\tau_c = 1/(2\pi f_c)$ is the filter parameter for a first-order lowpass critical frequency of f_c (Hz). The filter bandwidth is chosen to reduce noise from the human input data during teleoperation, since the observable data is generated by human motion. Ideally $f_c = 10$Hz, at the human reactive bandwidth as in Chapter 4. However due to sample rate restrictions ($\Delta = 0.04$s for the particle filter update), $f_c = 2$Hz, to achieve desired lowpass filter performance when using zero-order hold discretization.

Observation likelihood. The observation likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ is needed to perform the Bayesian update step of the particle filter, which adjusts weights on target hypotheses $\boldsymbol{x}_t \in \mathcal{X}_t$ based on observation feedback $\boldsymbol{z}_t$. The likelihood is expanded to

$$p(\boldsymbol{z}_t|\boldsymbol{x}_t) = p(\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \dot{\tilde{\boldsymbol{v}}}_h, \tilde{\boldsymbol{f}}|\boldsymbol{x}_t) \tag{A.6}$$

$$= p(\tilde{\boldsymbol{x}}_h|\boldsymbol{x}_t)p(\dot{\tilde{\boldsymbol{v}}}_h|\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \boldsymbol{x}_t)p(\tilde{\boldsymbol{f}}_h|\tilde{\boldsymbol{x}}_h, \boldsymbol{x}_t). \tag{A.7}$$

The elements of this expansion are computed according to the following.

- The position likelihood term $p(\tilde{\boldsymbol{x}}_h|\boldsymbol{x}_t) = \mathcal{N}(\tilde{\boldsymbol{x}}_h; \boldsymbol{x}_t, \boldsymbol{R}_x)$ weights likely targets within a normal probability distribution around the handle position $\tilde{\boldsymbol{x}}_h$. The covariance $\boldsymbol{R}_x$ is computed from the training data in Chapter 5 during the normalization step during data pre-processing. This element of the likelihood constrains the target search space to occur near the handle position $\boldsymbol{x}_h$, which improves particle filter performance, especially during re-sampling. However, this restricts the likelihood validity to regions near the task, making the likelihood invalid during transitions between holes. This is reasonable since the user is asked to press the button on the haptic device during the cleaning operation, and the particle filter updates can be restricted to these in-task events. In general, if this explicit communication is not available, another method for inferring an in-task versus out-of-task condition is needed.

- The acceleration likelihood term $p(\dot{\tilde{\boldsymbol{v}}}_h|\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \boldsymbol{x}_t)$ weights likely accelerations from the model Ξ learned in Chapter 5. This element provides observability for which component

of the task the user is demonstrating, helping to indicate where the system is w.r.t. the model center $\boldsymbol{x}_t$. This is particularly useful for identifying the target location in $\hat{x}$, the principal direction of stroke during cleaning. Because the acceleration data is noisy, this likelihood generalizes well to new users since it captures general trends in the motion. The likelihood is computed via the following. For each action manifold $a \in \mathcal{A}$ in the model Ξ, the Gaussian likelihood

$$p(\dot{\tilde{\boldsymbol{v}}}_h|\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \boldsymbol{x}_t; a) = \mathcal{N}(\dot{\tilde{\boldsymbol{v}}}_h; \boldsymbol{h}(\hat{\boldsymbol{\chi}}; a), \boldsymbol{R}(\hat{\boldsymbol{\chi}}; a)) \tag{A.8}$$

where $\boldsymbol{h}(\hat{\boldsymbol{\chi}}; a)$ is the same as in (5.14) and

$$\boldsymbol{R}(\hat{\boldsymbol{\chi}}; a) = \bar{\boldsymbol{\Sigma}}_{\dot{v}}(\hat{\boldsymbol{\chi}}; a) + \boldsymbol{R}_{\dot{v}}, \tag{A.9}$$

for $\bar{\boldsymbol{\Sigma}}_{\dot{v}}(\hat{\boldsymbol{\chi}}; a)$ also defined in (5.14), and an acceleration measurement covariance $\boldsymbol{R}_{\dot{v}}$ determined during the data pre-filtering step in Chapter 5. The system state

$$\hat{\boldsymbol{\chi}} = \begin{pmatrix} \tilde{\boldsymbol{x}}_h - \boldsymbol{x}_t \\ \tilde{\boldsymbol{v}}_h \end{pmatrix} \tag{A.10}$$

normalizes the position data to the model reference frame. The likelihood is then the maximum from the set

$$p(\dot{\tilde{\boldsymbol{v}}}_h|\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \boldsymbol{x}_t) = \max_a \; p(\dot{\tilde{\boldsymbol{v}}}_h|\tilde{\boldsymbol{x}}_h, \tilde{\boldsymbol{v}}_h, \boldsymbol{x}_t; a). \tag{A.11}$$

- Lastly, the force likelihood term $p(\tilde{\boldsymbol{f}}_h|\tilde{\boldsymbol{x}}_h, \boldsymbol{x}_t)$ weights likely force measurements. While the previous likelihoods are sufficient to localize the target in the $\hat{x}$ direction, this term is needed to find the hole location with sufficient accuracy in the $\hat{y}$ direction. However, because there is little motion in the $\hat{y}$ direction during task demonstrations for training, the model Ξ learned in Chapter 5 does not provide sufficient information to address this uncertainty. Specifically, haptic operation exhibits deformations in $\hat{y}$ outside of the kinesthetic demonstration domain, resulting in poor force estimates from the model Ξ in these regions.

To address this generalization issue, an empirical brush stiffness model provides the needed accuracy for force prediction in $\hat{y}$. Expected forces in $\hat{y}$ are

$$\mathbb{E}\left[\tilde{f}_{\hat{y}}\right] = \begin{cases} -K_{brush}(d_{\hat{x}}) \cdot d_{\hat{y}} & d > 0 \\ 0 & d \leq 0 \end{cases} \tag{A.12}$$

where $d_{\hat{x}} = \tilde{x}_{h,\hat{x}} - x_{t,\hat{x}}$ is the brush insertion depth (negative indicates the brush is not inserted), $d_{\hat{y}} = \tilde{x}_{h,\hat{y}} - x_{t,\hat{y}}$ is the brush displacement from equilibrium, and the stiffness $K_{brush}(d_{\hat{x}})$ is an affine model

$$K_{brush}(d_{\hat{x}}) = M_b \cdot d_{\hat{x}} + B_b. \tag{A.13}$$

The parameters M_b, B_b are found empirically from a brush stiffness test. In this test, the brush is deflected in $\hat{y}$ at various depths of insertion d. Each insertion depth produces a stiffness estimate from Hooke's law. Least squares over these values produce the parameter estimates. Figure A.3 shows the test data to produce this empirical model with the final model prediction in (A.12), and Figure A.4 shows the stiffness model in (A.13).

A.2.1 Study on the likelihood elements

Different elements of the likelihood are needed because they each include unique information to improve the observability of the target location. To illustrate the unique contribution of each element, different combinations were evaluated on a set of four teleoperation sequence demonstrations from the user (i.e. the teleoperation (TO) mode of the experiments in Section 6.3). The estimator was seeded with the same initial conditions for each evaluation to promote a fair comparison.

Evaluation metrics

The following metrics are examined in the evaluation.

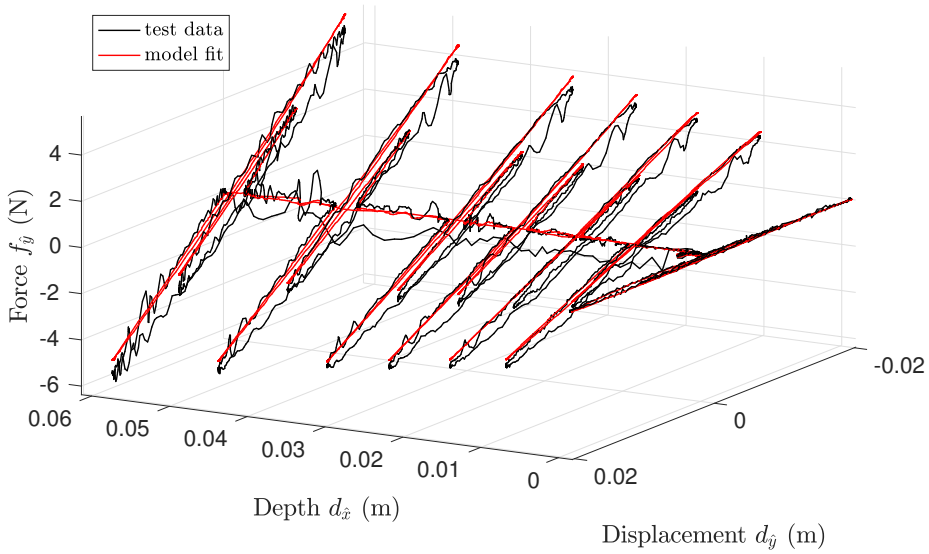


Figure A.3: The brush stiffness test captures brush stiffness in $\hat{y}$ at different insertion depths $d_{\hat{x}}$. The empirical model prediction closely matches the data.

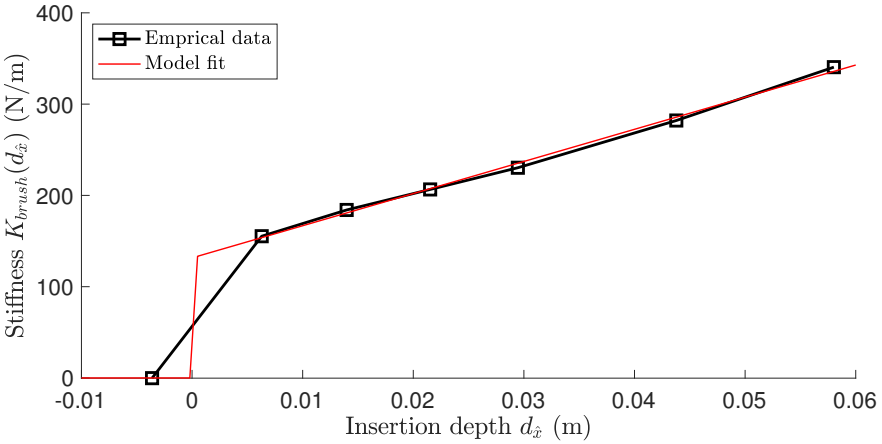


Figure A.4: The brush stiffness model shows a piece-wise linear change in brush stiffness as a function of insertion. The piece-wise elements capture the stiffness tendency $K_{brush} \to 0$ for $d_{\hat{x}} \leq 0$.

- **MSE** (mean squared error). This quadratic error metric considers the difference between estimate and ground truth. Ideally, the estimator MSE is low (i.e. perfect = 0). This evaluation considers the MSE in independent axes, $\hat{x}$ (denoted **MSE-X**) and $\hat{y}$ (**MSE-Y**), since different likelihood combinations affect these axes independently. For example, the MSE for $\hat{x}$ is

$$\textbf{MSE-X} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbb{E}[x^i_{t,\hat{x}}] - x^i_{t,\hat{x}}\|_2^2 \tag{A.14}$$

  where i is the sample, n is the number of samples in the test, $x^i_{t,\hat{x}}$ is the true target location in the $\hat{x}$ direction, and $\mathbb{E}[x^i_{t,\hat{x}}]$ is the expected value (i.e. from the estimator) of the target location in the $\hat{x}$ direction.

- **ANEES** (average normalized estimator error squared). This metric, defined in (3.18), is a composite metric that considers both the estimate error and covariance. For example, a large error in the estimate may exist (i.e. MSE >> 0), but if the estimator includes a large covariance to indicate the uncertainty in this estimate, then the *density*, i.e. $p(x_t)$, is trustworthy. A high value of ANEES (i.e. >> 1) indicates an severely overconfident estimator, with missing information or uncertainties that are too small in the likelihood. A low value of ANEES (i.e. < 1) indicates an underconfident estimator, i.e. with conservative uncertainty in the likelihood. An ideal estimator has an ANEES = 1. Like MSE, this metric is an average of the estimator performance for the entire test time, i.e. $i = \{1, \ldots, n\}$.

- **AE** (absolute error). This metric is the absolute error between the estimate and ground truth. Like MSE, this evaluation considers error in independent axes $\hat{x}$ (denoted **AE-X**) and $\hat{y}$ (**AE-Y**). However, this metric is evaluated at the index when the user specifies the first hole cleaning is complete, i.e. through release of the haptic device button. This is the point when automation will evaluate quality of estimate to either

take control or leave control with the human. For example, the AE for $\hat{x}$ is

$$\textbf{AE-X} = |\mathbb{E}[x_{t,\hat{x}}^{i^*}] - x_{t,\hat{x}}^{i^*}| \tag{A.15}$$

where i^* is the sample index at the first button release, $x_{t,\hat{x}}^{i^*}$ is the true target location in the $\hat{x}$ direction, and $\mathbb{E}[x_{t,\hat{x}}^{i^*}]$ is the expected value (i.e. from the estimator) of the target location in the $\hat{x}$ direction.

- **Conf** (confidence). This metric is the confidence in the estimate, i.e. from (3.15), at the first button release i^*, i.e.

$$conf = 1 - \sum_{k=1}^{N} w^{(k)} \mathbb{I}(\boldsymbol{x}_t^{(k)}) \tag{A.16}$$

where $k = \{1, \ldots, N\}$ is the particle index, $w^{(k)}$ is the particle weight, and $\mathbb{I}(\boldsymbol{x}_t^{(k)})$ is a boolean mask indicating if particles are inside or outside the collision region. Note that $conf = 1 - \Pr(\mathcal{C})$ since the estimator confidence is $\neg \Pr(\mathcal{C})$, i.e. the probability of colliding with the coupon around the hole. Since a sufficiently high confidence, i.e. $conf > P^*$, is required for the automation to take control of the robot and complete the task, higher confidence at the end of the first hole is desired.

Independent variables

The following likelihoods are evaluated. Note that conditional dependencies are omitted for brevity.

- **X12.** This likelihood is solely the position likelihood term $p(\boldsymbol{z}_t|\boldsymbol{x}_t) = p(\tilde{\boldsymbol{x}}_h)$ where $\boldsymbol{x}_h$ is position in $\hat{x}$ and $\hat{y}$.

- **X1A.** This likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t) = p(\tilde{\boldsymbol{x}}_{h,\hat{x}})p(\dot{\tilde{\boldsymbol{v}}}_{h,\hat{x}})$ considers the position likelihood in $\hat{x}$ as well as the acceleration $\dot{v}$ in $\hat{x}$. Recall that $\hat{x}$ is the principal direction of motion, and accelerations in $\hat{y}$ are predominantly noise.

	Likelihood $p(\cdot)$				*Evaluation Results*					
					Overall			After completing hole 1		
					ANEES	**MSE-X**	**MSE-Y**	**Conf**	**AE-X**	**AE-Y**
	$x_{\hat{x}}$	$x_{\hat{y}}$	$\dot{v}_{\hat{x}}$	$f_{\hat{y}}$	-	mm^2	mm^2	%	mm	mm
X12	x	x			1166±194	147±53	239±30	100±0	8.4±5.5	3.6±0.4
X1A	x		x		4±2	76±6	273±67	75±0.1	2.8±3.3	1.4±0.2
X1F	x			x	114±78	146±36	240±37	100 ±0	8.6±4.6	2.0±0.3
X1AF	x		x	x	19±15	80±8	234±32	99.9±1	2.9±2.0	1.4±0.5

Table A.1: Estimator likelihood evaluation results.

- **X1F.** This likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t) = p(\tilde{\boldsymbol{x}}_{h,\hat{x}})p(\tilde{\boldsymbol{f}}_{h,\hat{y}})$ considers the position likelihood in $\hat{x}$ as well as the force f in $\hat{y}$. Recall that the likelihood $p(\tilde{\boldsymbol{f}}_h)$ predicts only side forces in $\hat{y}$.

- **X1AF.** This likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t) = p(\tilde{\boldsymbol{x}}_{h,\hat{x}})p(\dot{\tilde{\boldsymbol{v}}}_{h,\hat{x}})p(\tilde{\boldsymbol{f}}_{h,\hat{y}})$ considers the position likelihood in $\hat{x}$, acceleration $\dot{v}$ in $\hat{x}$, and force f in $\hat{y}$.

A.2.2 Results and discussion

Results of the evaluation are presented in Table A.1. Estimate error and variance at the end of cleaning the first hole for each of the test cases is shown in Figure A.5.

Position information in isolation results in estimator bias.

Examining the results for **X12** in Table A.1, the *ANEES* is large (i.e. 1166), indicating the estimator is severely overconfident. If used for shared autonomy, at the end of hole $l = 1$, the automation would be 100% confident in an estimate that could be biased by 8.4 mm in $\hat{x}$ and 3.6 mm in $\hat{y}$. Note that biases in $\hat{y} > 3$ mm result in the robot striking the coupon, meaning this estimator would not be successful.

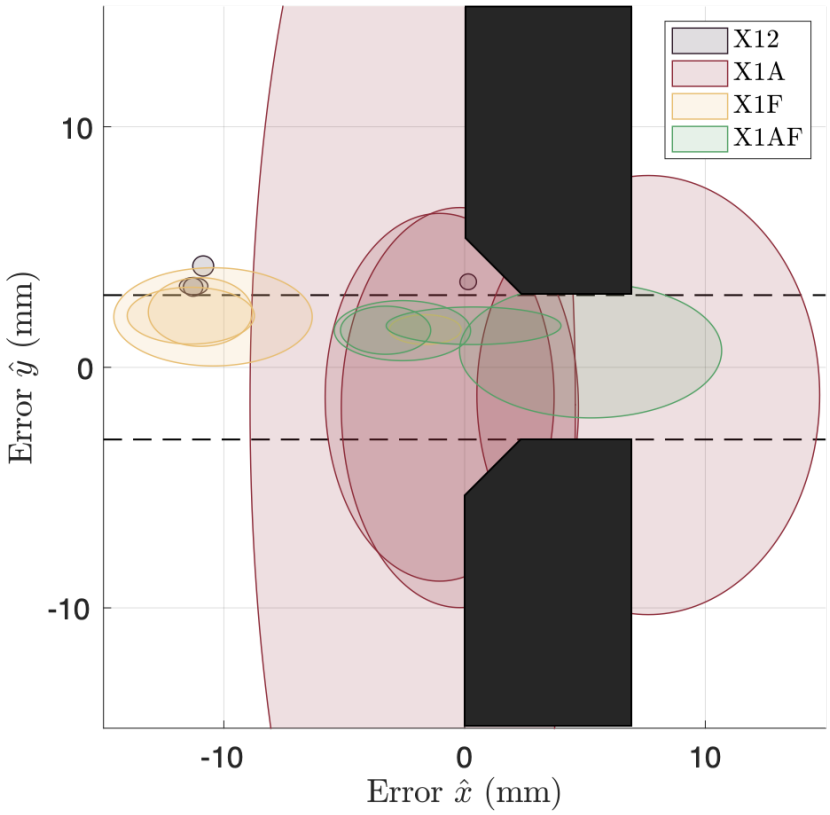


Figure A.5: Target estimates and variances (depicted as ellipses) are shown at the end of teleoperation for hole $l = 1$. Note that there are four variance ellipses for each likelihood case since the same four datasets were evaluated for each case.

To understand why this is the case, examine the error ellipses for **X12** in Figure A.5. As the particle filter runs, position updates at each time step continue to collapse the uncertainty in the error estimate, resulting in small estimates at the end of hole $l = 1$. However, these estimates tend to be biased positive in $\hat{y}$ and negative $\hat{x}$.

The positive bias in $\hat{y}$ is a result of the *teleoperator* being biased in $\hat{y}$ during the cleaning. Generally, since the teleoperator feedback in this non-principal axis of motion could be biased larger than the tolerance required for successful execution, using operator position in $\hat{y}$ is

not acceptable to yield good estimator performance. For this reason, remaining likelihood evaluations do not include updates from handle position in $\hat{y}$.

The negative bias in $\hat{x}$ is a result of early button presses that bias the likelihood to regions before cleaning begins. For this reason, motion, i.e. the accelerations during cleaning, are needed to provide additional insight to the cleaning operation itself.

Motion alone is insufficient for localization.

Examining the results for **X1A** in Table A.1, the *ANEES* is small (i.e. 4), indicating the estimator confidence is substantially more accurate than **X12**. While the average errors *AE* at the end of the hole cleaning are low, the estimator confidence *conf* is also much lower at 75%. This is a result of uncertainty in the $\hat{y}$ direction. The estimator indicates that observability in $\hat{y}$ is poor with this likelihood scheme, evidenced by high *MSE-Y*.

This uncertainty is obvious in error ellipses for **X1A** in Figure A.5. The variances are large in $\hat{y}$, indicating the estimator does not have sufficient information to close the uncertainty in this axis from this likelihood scheme. For this reason, the forces resulting from the interaction between the robot and the workpiece are needed to provide observability in $\hat{y}$.

Both interaction forces and motion (task dynamics) are needed for localization.

While kinetics information improves the uncertainty in $\hat{y}$, it is biased in $\hat{x}$, evidenced by the error ellipses for **X1F** in Figure A.5. Without motion information, this estimate suffers from the uncertainty collapse of position updates. However, this has a secondary affect: since the interaction model is dependent on the position in $\hat{x}$ to predict brush stiffness, this bias affects the equilibrium prediction in $\hat{y}$ as well. For this reason, both task motions and interaction forces, i.e. the *full task dynamics* are needed for sufficient localization.

Results for the selected estimator likelihood **X1AF** in Table A.1 indicate an estimator with high confidence *conf* in the estimate (99.9%), low absolute errors *AE-X, AE-Y* $(2.9, 1.4$ mm), and a relatively low *ANEES* of 19. The error ellipses in Figure A.5 confirm this story, indicating after one hole clean, the estimator attains a sufficient estimate for automation.

As automation continues to perform the task, the estimator updates can continue, especially to improve the target estimate in $\hat{y}$ from interaction forces.

Study summary.

These results illustrate the need for task dynamics to localize from data acquired during manual operation. Two fundamentally different models are needed for the localization: (i) human accelerations are used to localize the target in the principal axis of motion; and (ii) interaction kinetics that are decoupled from human motion due to high noise in the human input are needed to localize the target in the non-principal axis of motion.

A.2.3 Other filter information

Particle filter parameters. The number of particles $n_p = 2000$ is chosen to achieve sufficient coverage of the posterior density. Systematic re-sampling is performed when the number of effective particles $\hat{N}_{eff}$ is less than $n_c = 400$ (20% of the total number of particles). The number of effective particles

$$\hat{N}_{eff} = \sum_{p=1}^{n_p} \frac{1}{w_p^2} \tag{A.17}$$

where w_p is the weight of particle p. The confidence in automation is computed as in Chapter 3, integrating the particle weights over the collision geometry of the region around the hole. A critical threshold of $P^* = 0.997$ (3σ bounds) is used to determine sufficient confidence in the localization estimate for the automation to complete the remaining holes in the sequence.

Figure A.6 shows a typical estimation error profile for a trial in an evaluation dataset of the cleaning sequence of holes 1-9, performed with haptic teleoperation. Discontinuities in the particle density occur during particle re-sampling. Note that the error falls within acceptable bounds by the time sufficient confidence is reached ($t = 8$s). Because this occurs during the first hole, the remaining 8 holes could be automated by the robot. Note that errors tend to be positive for the first few holes and negative for the last few holes. This is primarily

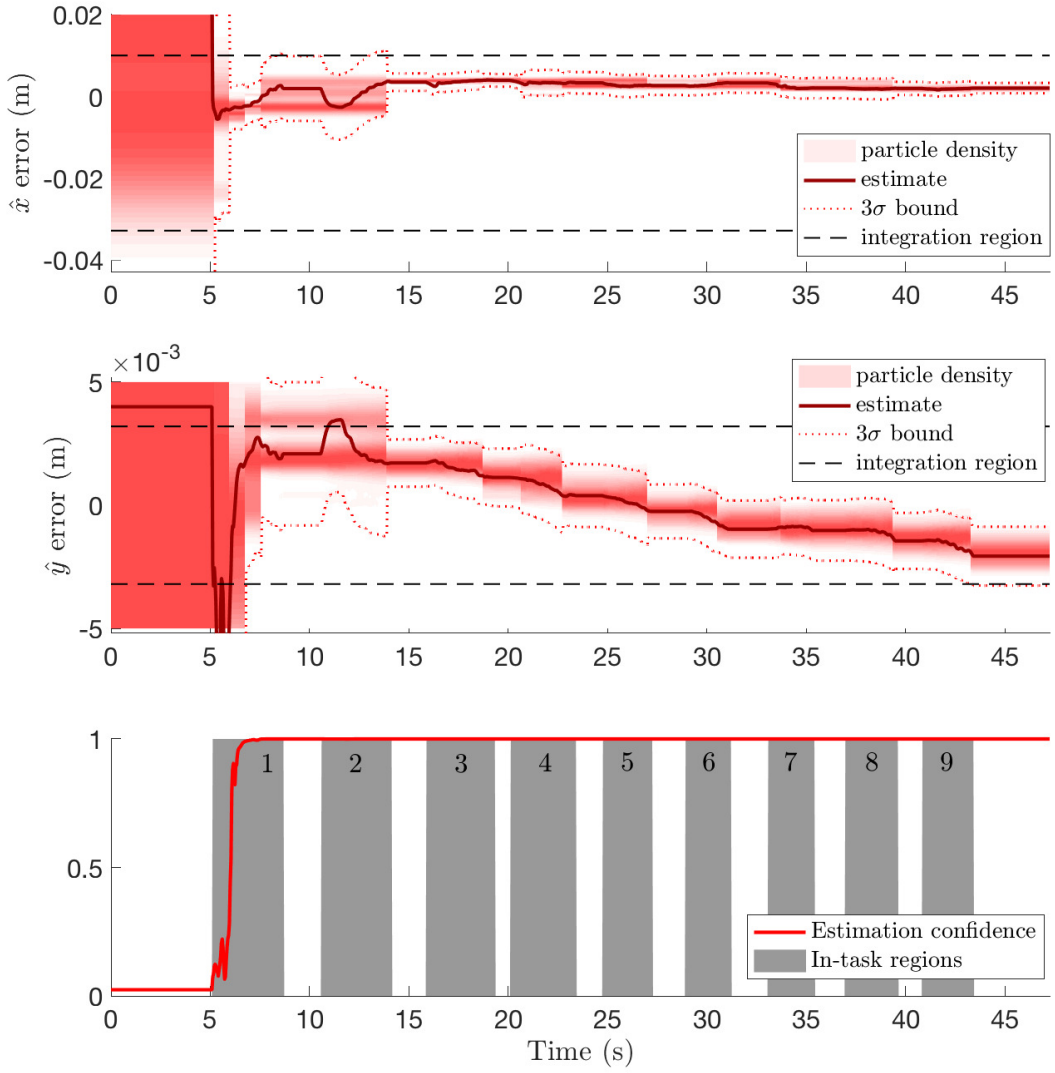


Figure A.6: A typical estimator profile shows target localization particle density and estimate errors. Confidence integration bounds are shown for reference. The estimation confidence is computed by integrating the weighted particle density over the integration bounds. Hole numbers are shown for reference.

due to a small orientation error of the robot base w.r.t. the coupon, making the assumption about the relationship between holes in the sequence slightly erroneous. However, because the integration bounds shown are conservative, this falls within acceptable levels for the system.

Hole location calibration. The particle filter for target localization assumes that the hole locations are known relative to each other. In practice, this is known via the computer aided design (CAD) model. However for the experimental setup, these values needed to be measured and the test setup calibrated so that a ground truth could be known. To determine this relationship, the distance between hole locations was measured using a set of dial calipers. The distances between holes were averaged and standard deviation computed from three measurements between each hole. The mean $\mu = 38.18$ mm, and standard deviation $\sigma = 0.012$ mm.

To verify the relationship between holes, and identify any small orientation issues, a simple test where the robot brush tip was dragged lightly along the surface of the test coupon in between holes was used to find the coupon surface. Data-points where the force sensor registered > 1 N force were marked as in contact with the surface.

The data is shown in Figure A.7. Curvature in the coupon is believed to be due to robot kinematic calibration errors, as the coupon itself is fairly straight when measured with a straightedge. This is plausible because the robot is custom assembled using M4/M5 bolts, with un-calibrated kinematics (kinematic values were ideal from design or found using dial calipers). Note that rotation offsets at the joints would introduce the curvature shown in the data. Calibration of the robot kinematics using a laser tracker or visual tracking would be useful to improve robot control and performance.

Two correction models for the relationship between hole locations were investigated (Figure A.7): orientation correction using a least squares affine transformation, and quadratic using a least squares polynominal. The orientation correction was used in experiments in Chapter 6 as this model achieved sufficient accuracy for successful operation.

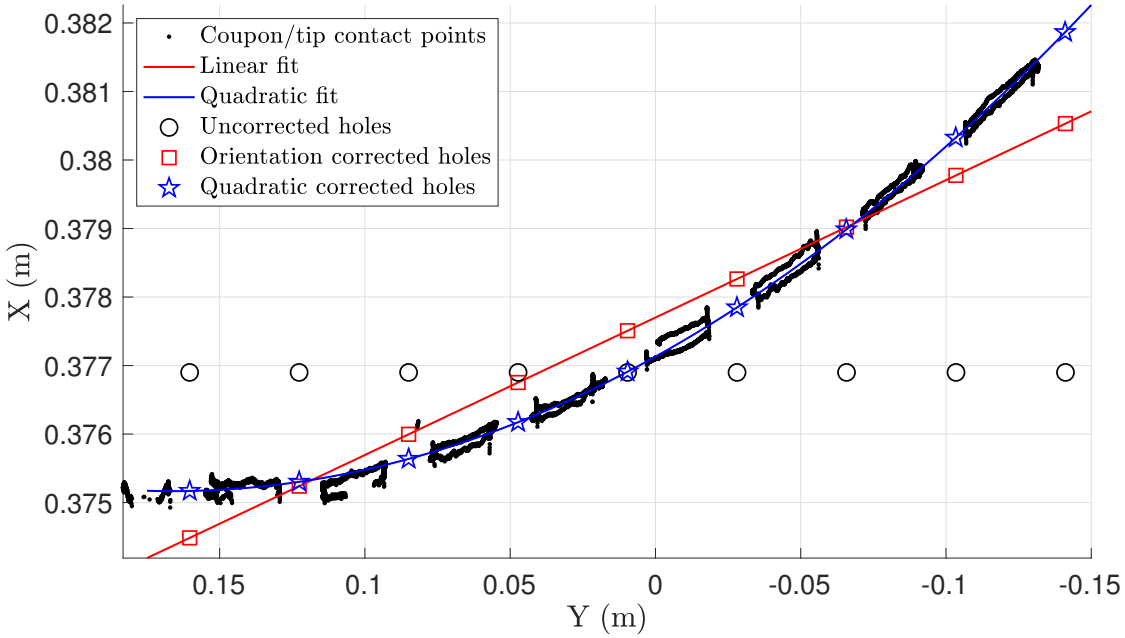


Figure A.7: Test coupon and hole relationship calibration data. Several models were considered to correct the original (uncorrected) hole location relationships. Ultimately, the linear orientation model was used to correct the original hole locations.

A.3 Off-Nominal Detection During Automation

It is possible for the localization converge to a poor estimate, especially during the resampling step where new particle distributions are randomly drawn around high-weight regions. In these cases, the automation will be overconfident in the estimate and can potentially try to perform the trajectory outside of the acceptable region for success. In this instance, the methods for off-nominal detection from Chapter 4 are useful to identify these cases and prevent the autonomy from causing damage.

The methods from Chapter 4 are modified to use the expected interaction forces between the brush and the workpiece to determine an off-nominal event. The mechanism is the same as in Section 4.2, however the expected force is computed from the task dynamics model

estimate $\boldsymbol{g}$ from (5.15) in Chapter 5, and the measured force $\boldsymbol{f}$ is from the sensor feedback.[1] This results in the likelihood in (4.12) being $\Pr(\boldsymbol{f}|\boldsymbol{g}, \eta)$. The mean $\breve{\boldsymbol{\mu}}$ and covariance $\breve{\boldsymbol{\Sigma}}$ in (4.13) are computed

$$\breve{\boldsymbol{\mu}} = \boldsymbol{g}(\boldsymbol{\chi}; a) \tag{A.18}$$

$$\breve{\boldsymbol{\Sigma}} = \boldsymbol{R}(\boldsymbol{\chi}; a) + \beta \boldsymbol{R}_f \tag{A.19}$$

where $\boldsymbol{R}(\boldsymbol{\chi}; a)$ is from (5.15), $\boldsymbol{R}_f$ is an additional sensor noise model, and β is a noise inflation factor. Typically, $\beta = 1$, however this parameter can be tuned in practice to $\beta > 1$ in order to prevent false positive off-nominal events from being detected and stopping the robot in nominal operation. For this experiment, $\beta = 1$. The measurement covariance $\boldsymbol{R}_f$ used is

$$\boldsymbol{R}_f = \begin{pmatrix} 30 & 0 \\ 0 & 20 \end{pmatrix} \tag{A.20}$$

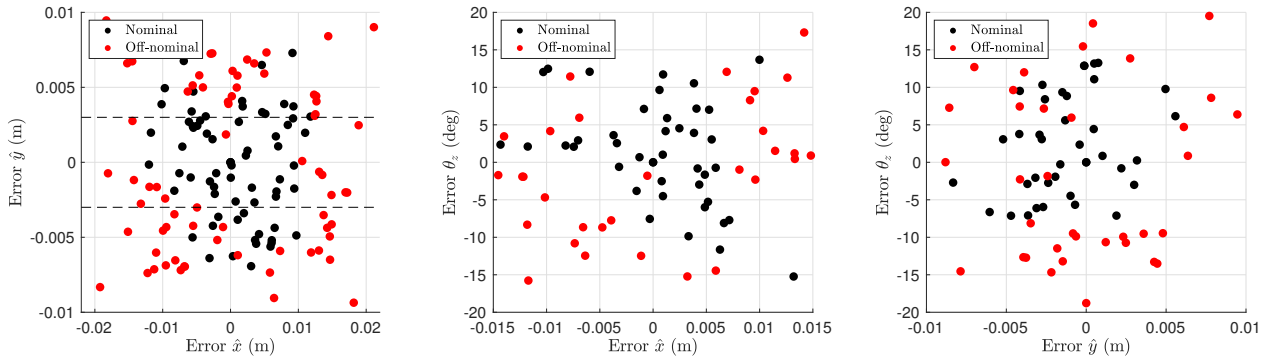
determined from the error between noisy sensor and prediction from the model learned in Chapter 5. Off-nominal events can be detected by comparing the probability of off-nominal to a critical threshold P^* (chosen as the 3σ value $P^* = 0.997$ for the experiments)

$$\Pr(\eta = \neg nom) > P^*. \tag{A.21}$$

Off-nominal experiments. To characterize the effect of localization errors on off-nominal events, a simple experiment was performed where artificial target position and orientation errors were introduced. In this experiment, the policy was run on hole 5 (used for training in Chapter 5. The orientation and translation errors represent movement and rotation of the coupon with respect to the robot. Orientation errors were applied at the center of the hole on the coupon.

Results of this experiment are shown in Figure A.8. In the translation test (Figure A.8a), the theoretical bounds from geometry in the $\hat{y}$ generally predict task success within errors

[1] Note these signals are filtered as in (4.10).



(a) Errors in $x - y$) (b) Errors in $x - \theta_z$ (c) Errors in $y - \theta_z$

Figure A.8: Results for off-nominal detection due to localization errors in $\hat{x}$, $\hat{y}$, and θ_z.

of ± 10 cm in $\hat{x}$. Outside of this region, off-nominal events are dominant. In the $x - \theta_z$ test (Figure A.8b), orientation errors of ± 10 deg are tolerable for errors in $7 > \hat{x} > -3$ cm. The result are asymmetric due to the robot tendency to "pull" towards negative $\hat{y}$ in operation. In the $y - \theta_z$ test (Figure A.8c), orientation errors of ± 10 deg are tolerable for errors in $5 > \hat{y} > -5$ cm. Note these results are also asymmetric due to the robot tendency to "pull" during accelerations.

Noise in the results is due to the nature of the test. In some cases, large orientation errors cause the brush to bend, and it must be manually straightened prior to the next test. Additionally, impact forces have potential to introduce bias into the force/torque sensor mid-test, which would increase false positive off-nominal detection rates.

Appendix B

EXPERIMENTAL RESULTS

This appendix includes data for the user study experiments in Chapter 6.3.

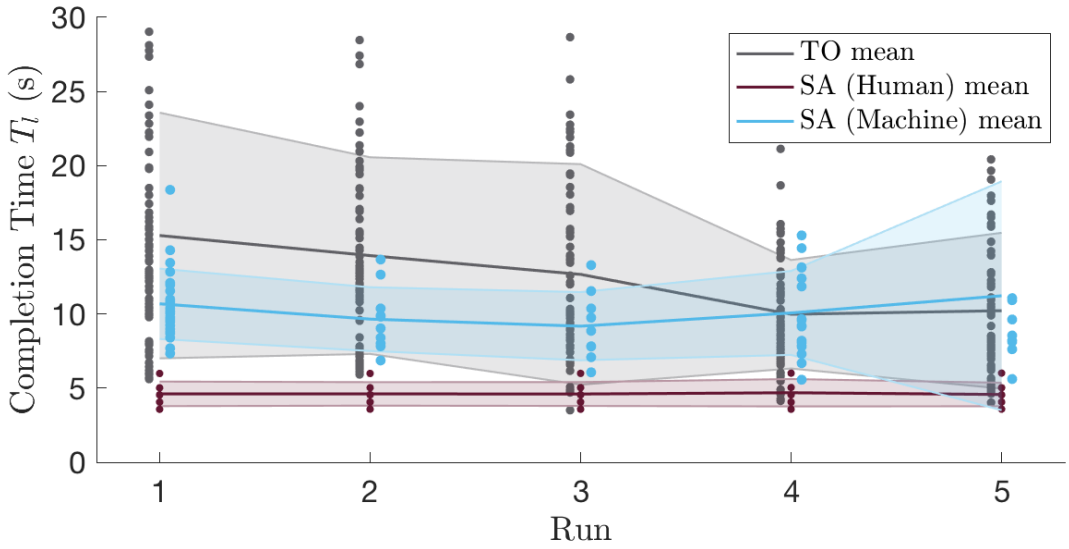


Figure B.1: Completion time over each run. Error bars denote standard deviation.

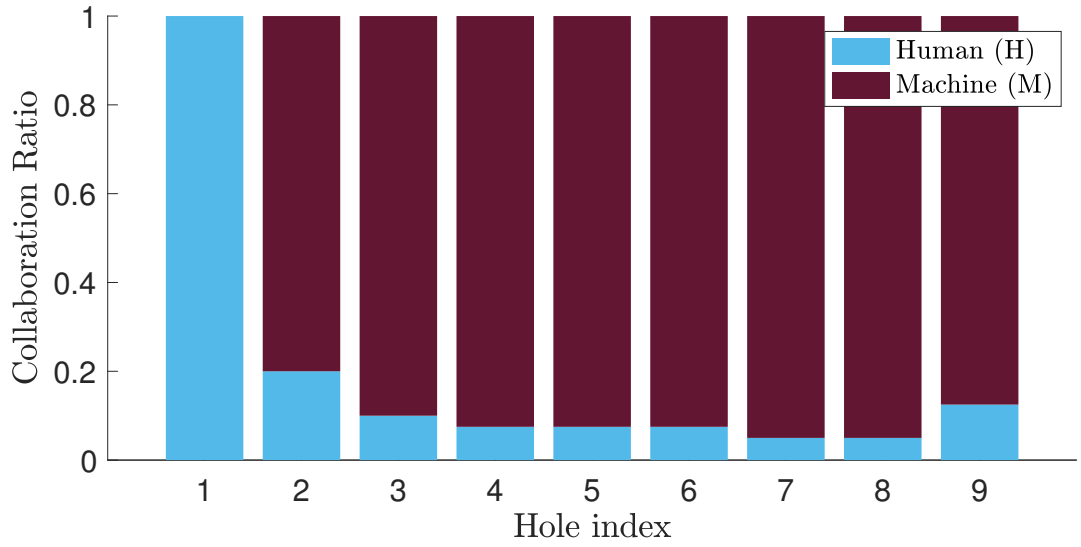


Figure B.2: Breakdown of the ratio of human (H) and machine (M) contribution for each hole during shared autonomy.

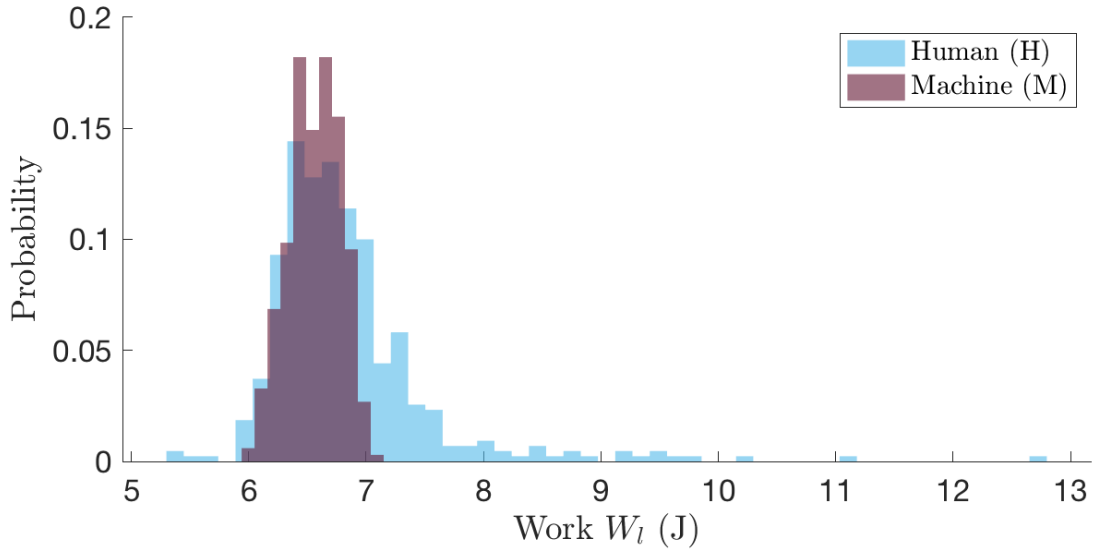


Figure B.3: Histogram of the work performed for each hole W_l during shared autonomy.

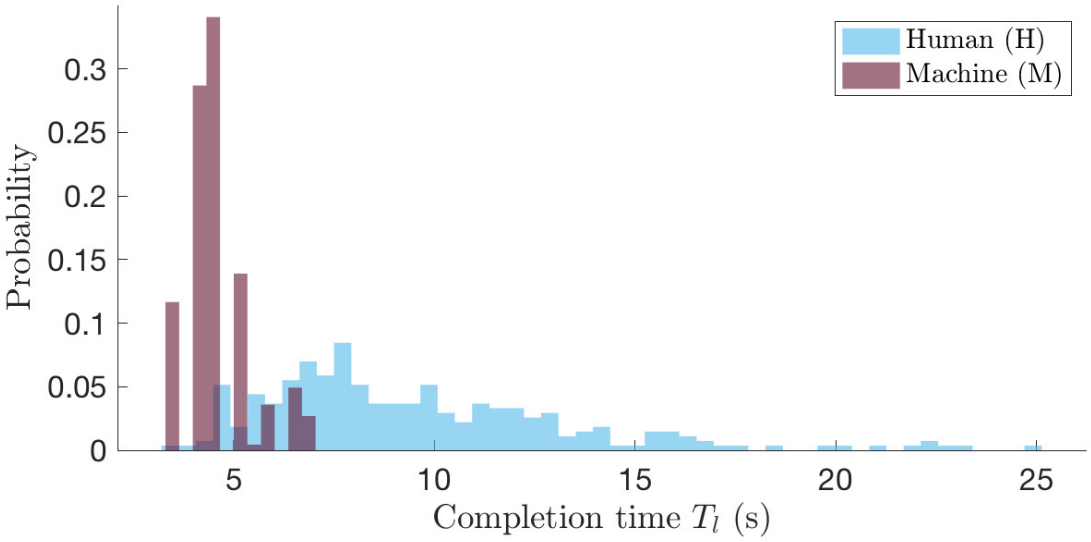


Figure B.4: Histogram of the completion time for each hole T_l during shared autonomy.

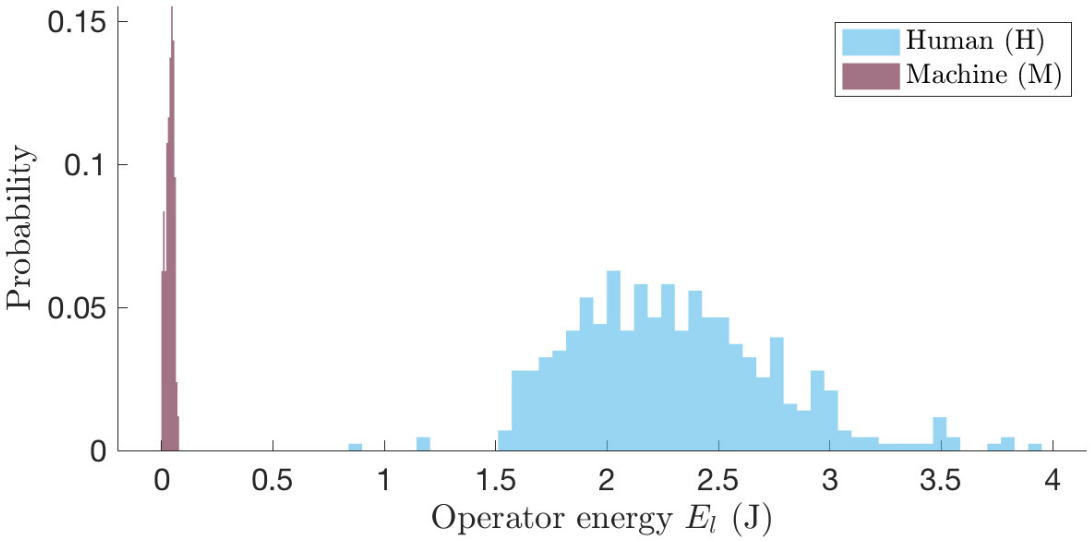


Figure B.5: Histogram of the expended operator energy for each hole E_l during shared autonomy.

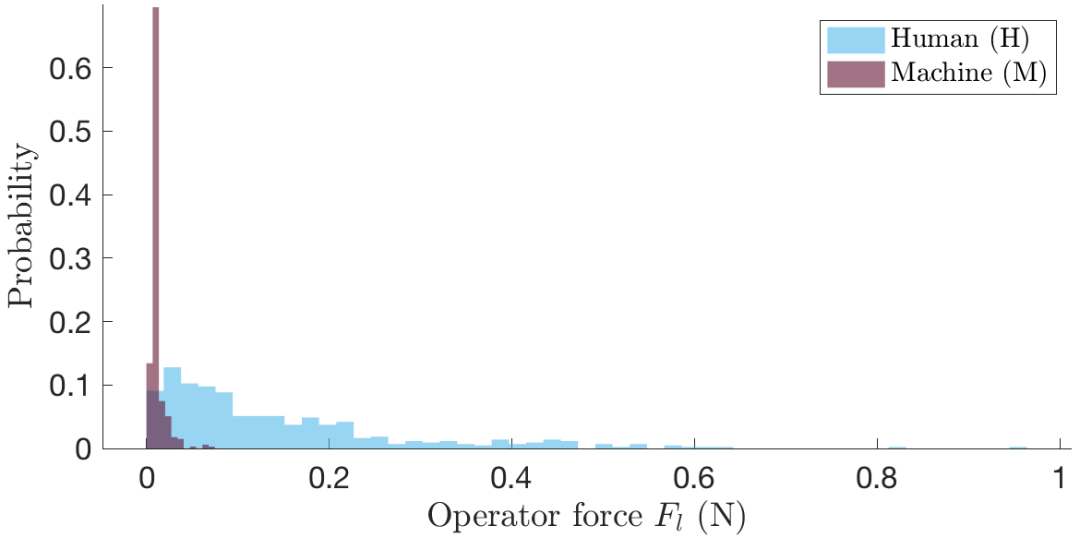
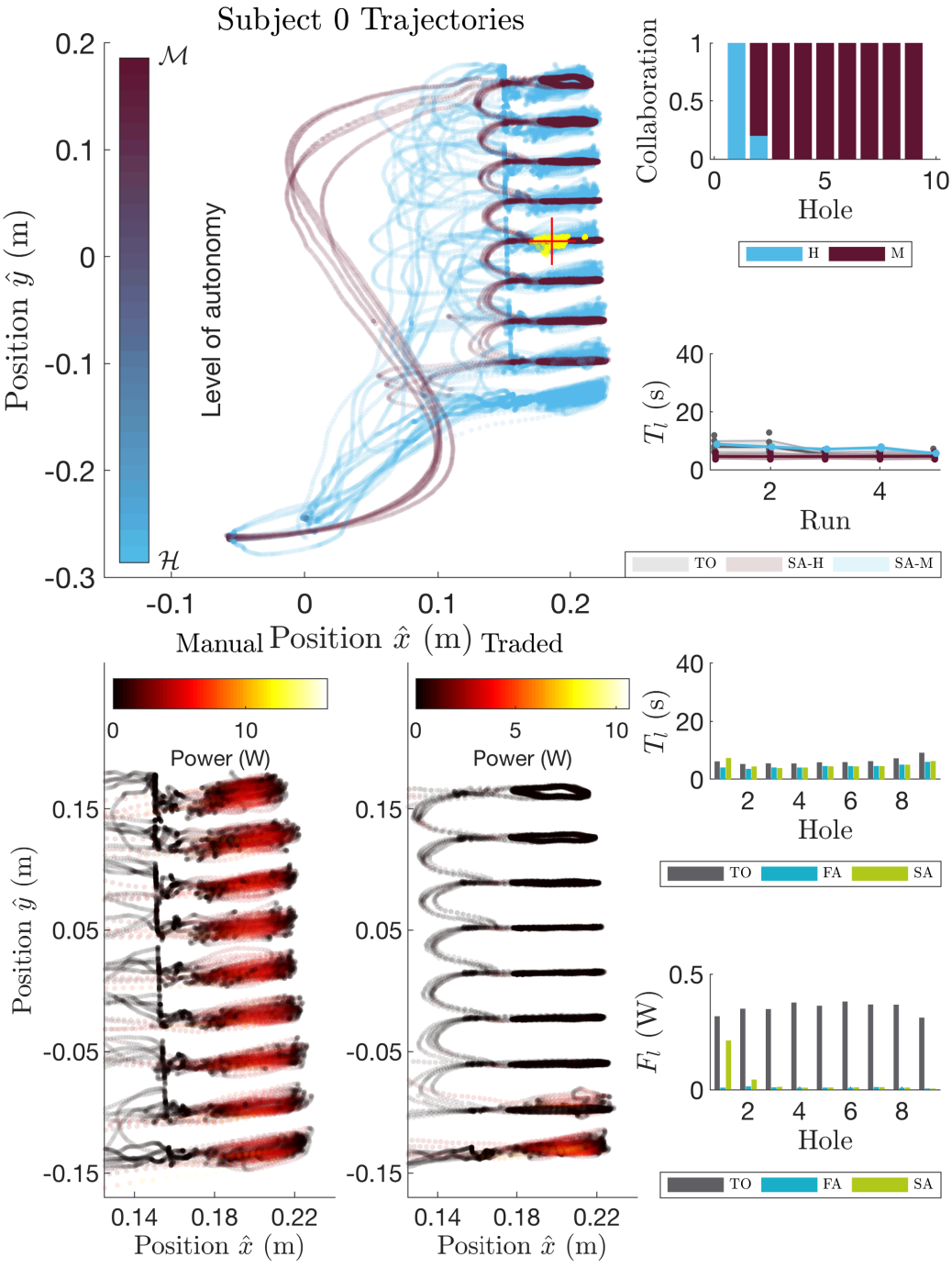
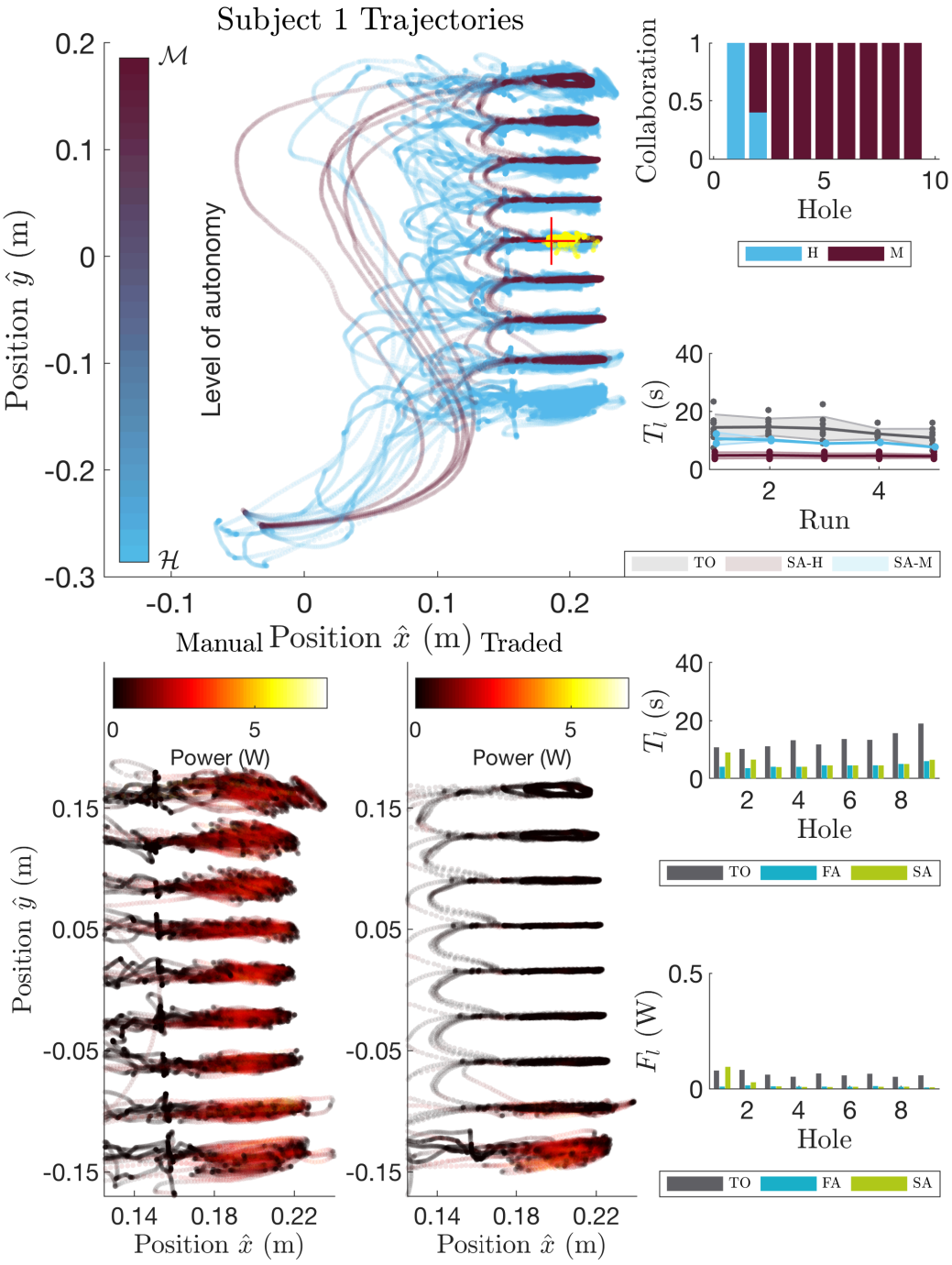
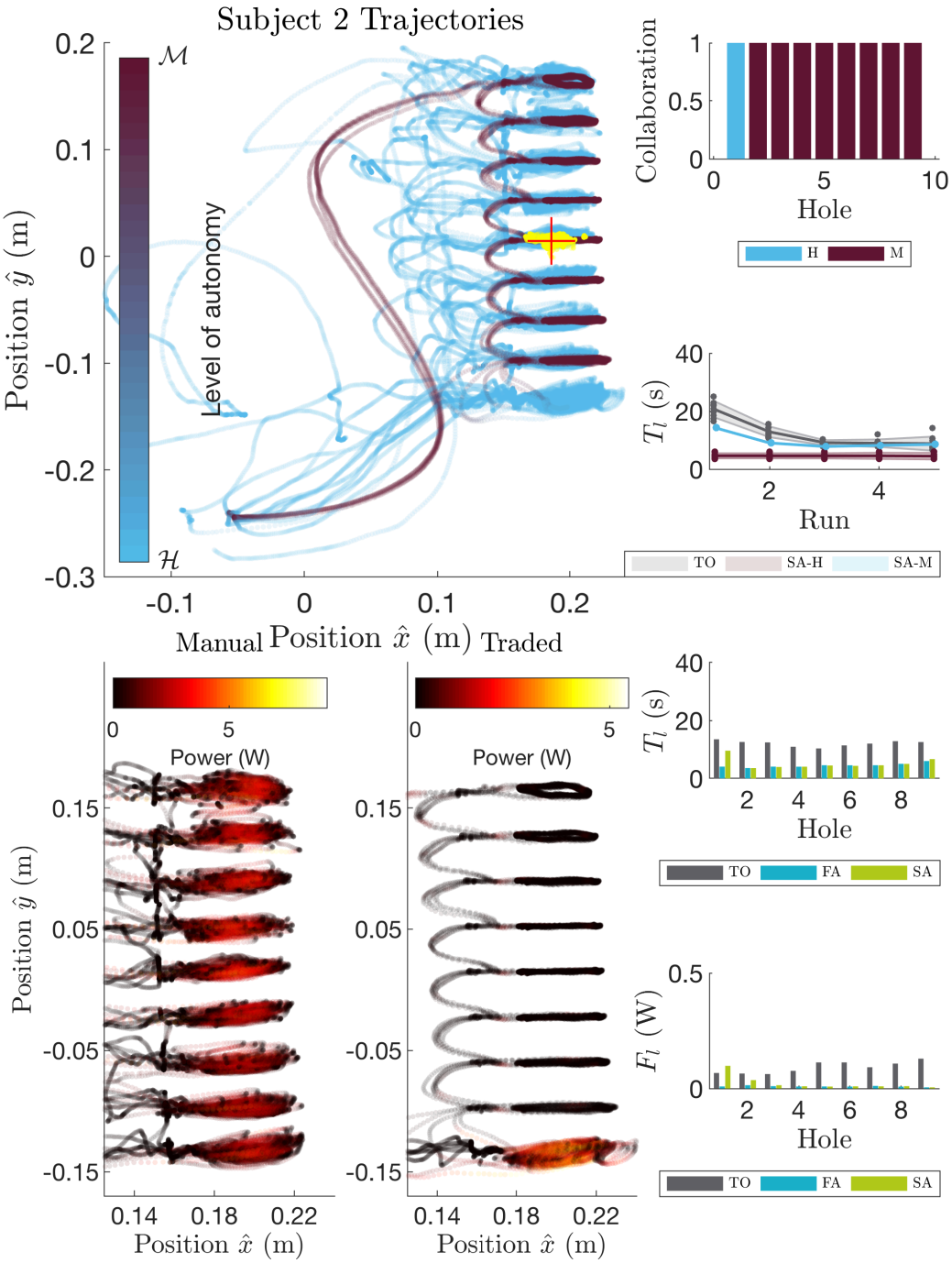
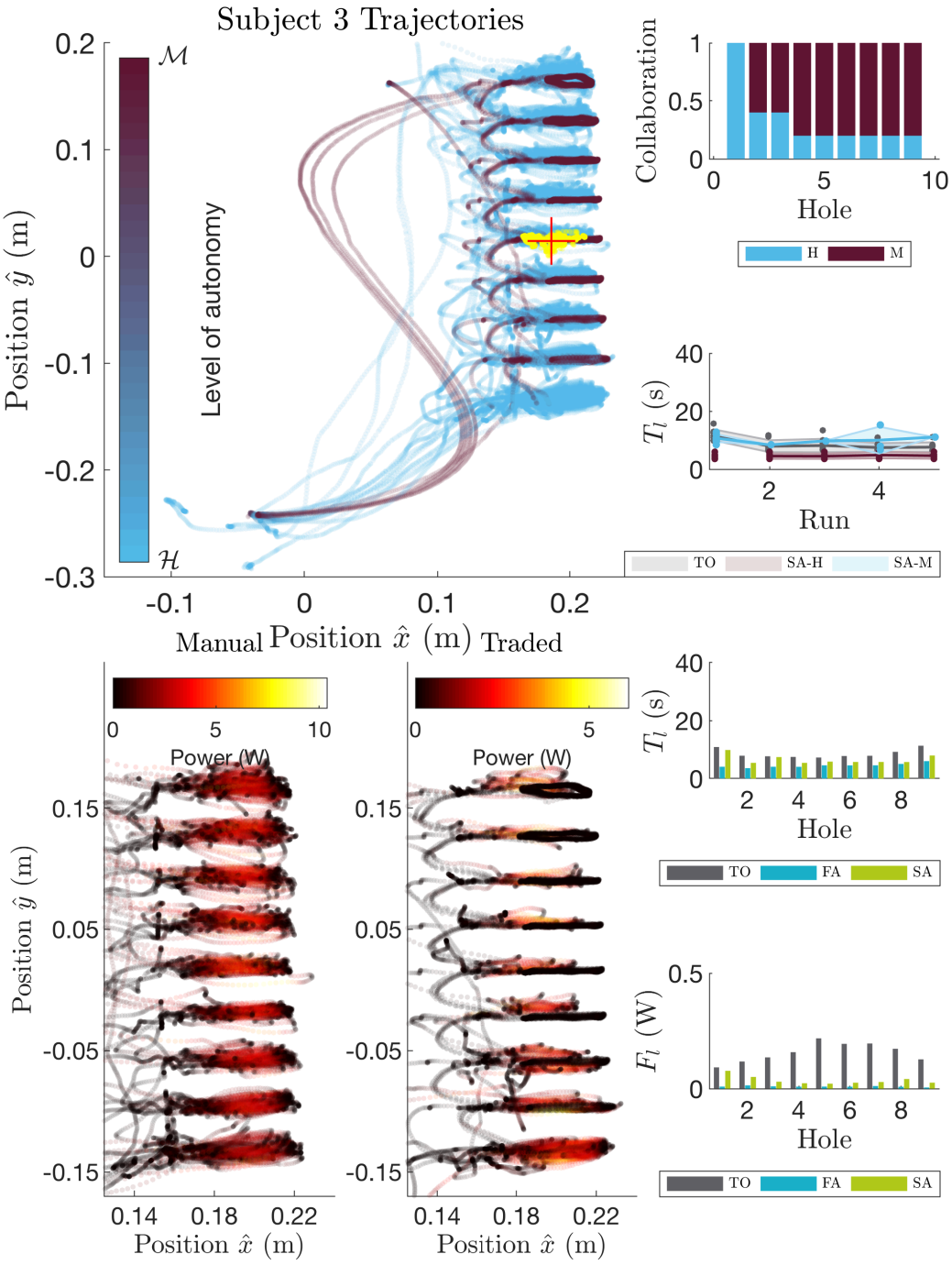


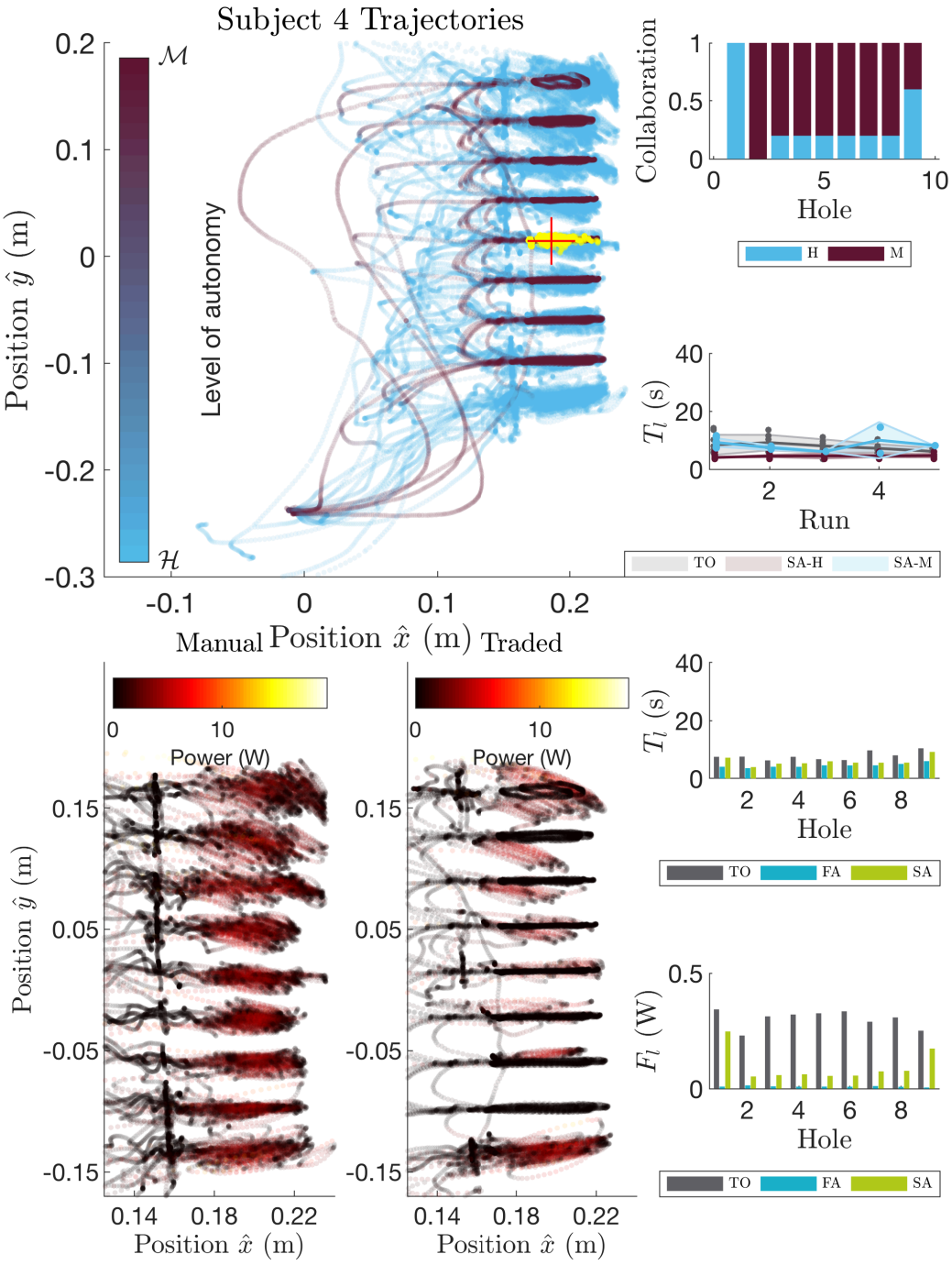
Figure B.6: Histogram of the RMS operator force for each hole F_l during shared autonomy.

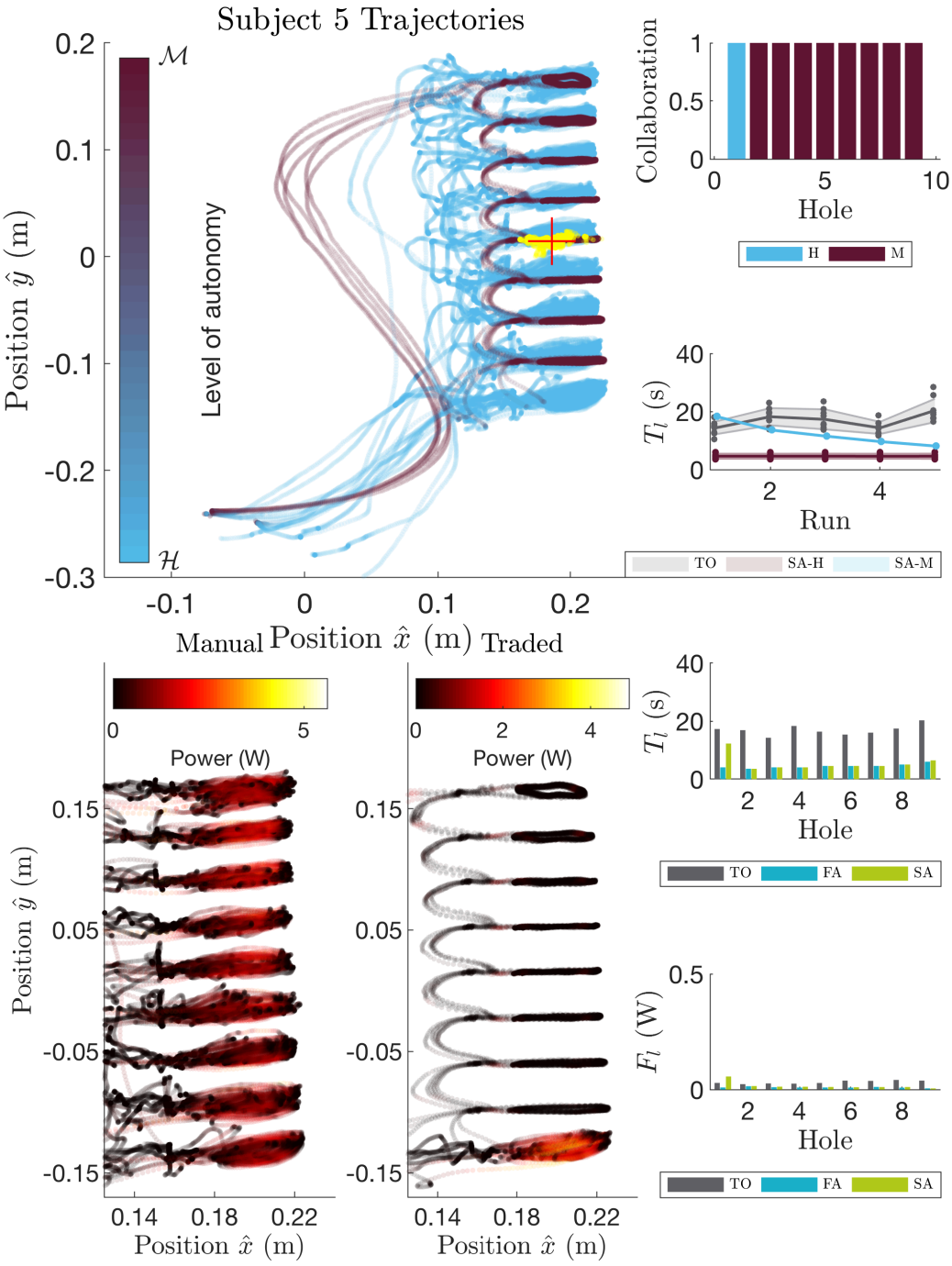

Subject 0 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)

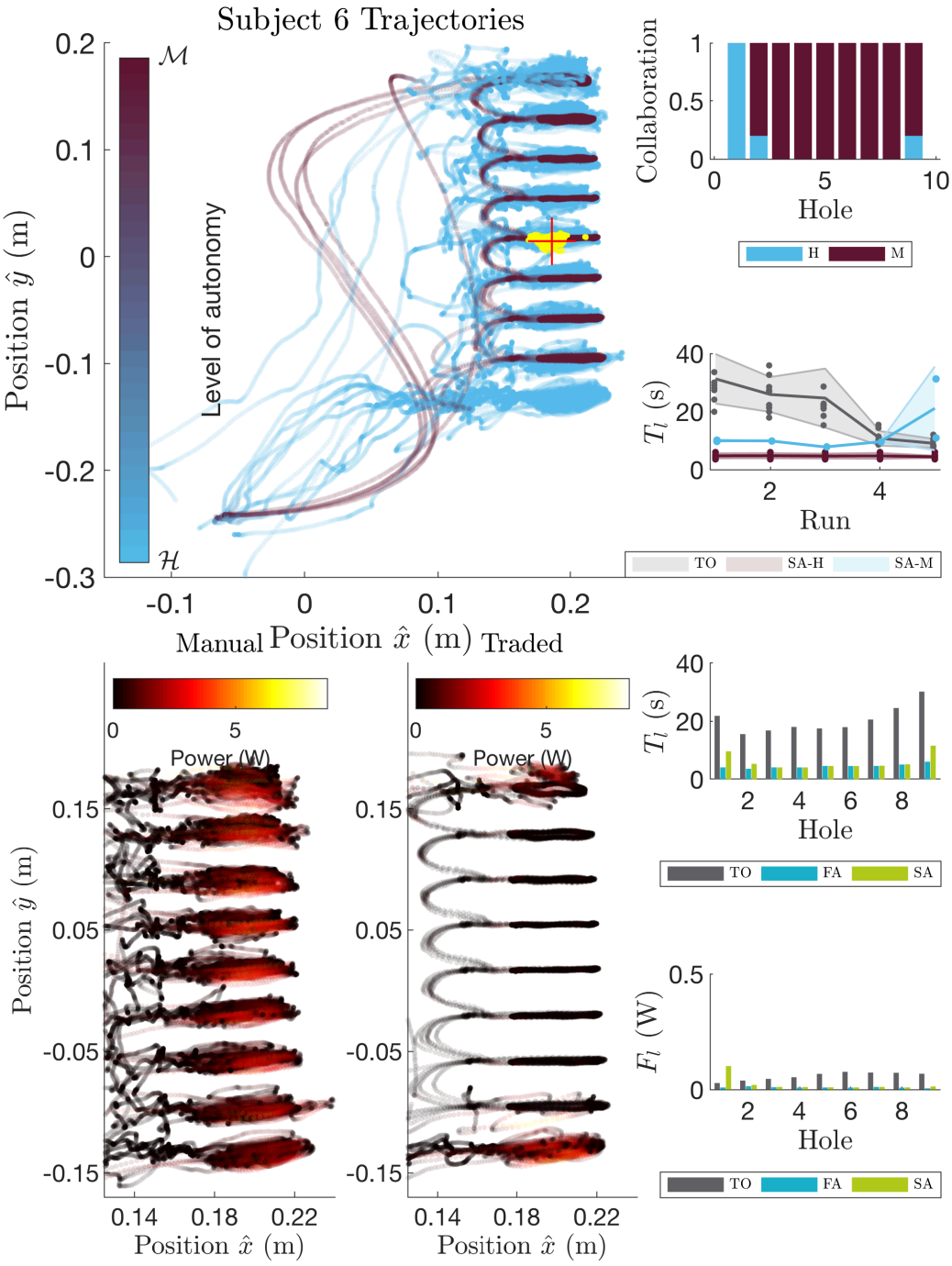

Subject 1 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)

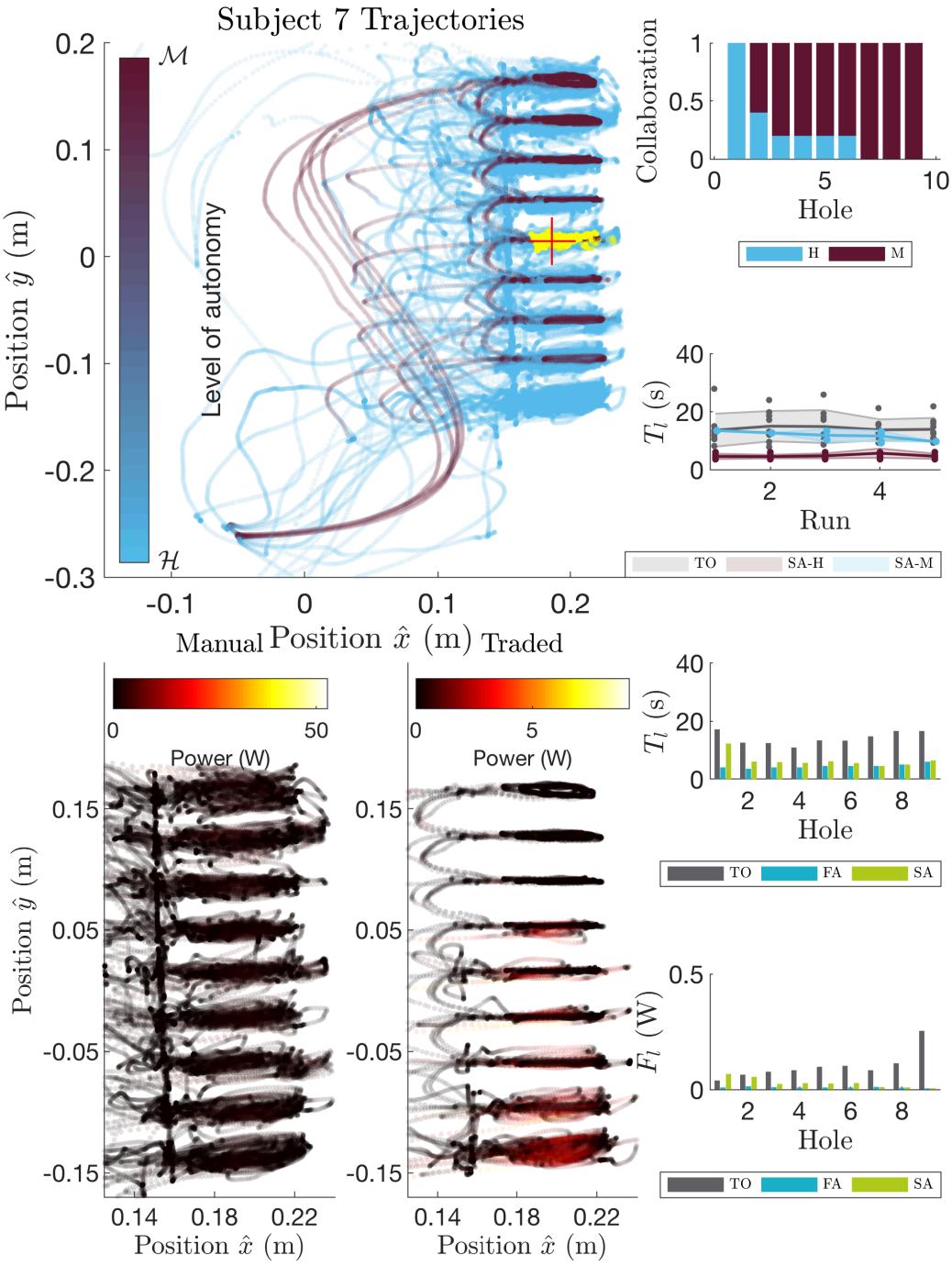

Subject 2 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)


Subject 3 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)


Subject 4 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)


Subject 5 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)


Subject 6 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)
Hole
TO
FA
SA


Subject 7 Trajectories
Level of autonomy
$\mathcal{M}$
$\mathcal{H}$
Position $\hat{y}$ (m)
Manual Position $\hat{x}$ (m) Traded
Collaboration
Hole
H
M
T_l (s)
Run
TO
SA-H
SA-M
Power (W)
Position $\hat{y}$ (m)
Position $\hat{x}$ (m)
T_l (s)
Hole
TO
FA
SA
F_l (W)

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files	Subject number
	#0

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	-1	0	**1**	2	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	1	**2**	3
2a	I felt uncomfortable with the robot (manual mode).	-3	**-2**	-1	0	1	2	3
2b	I felt uncomfortable with the robot (collaboration mode).	**-3**	-2	-1	0	1	2	3
3a	The robot was trustworthy (manual mode).	-3	-2	-1	**0**	1	2	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	0	1	**2**	3
4a	The robot was intelligent (manual mode).	-3	-2	**-1**	0	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	**1**	2	3
5a	I found what I was doing with the robot confusing (manual mode).	**-3**	-2	-1	0	1	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	**-3**	-2	-1	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	1	**2**	3
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	2	**3**
8	I prefered manual mode over collaboration mode.	**-3**	-2	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	-3	-2	-1	0	1	2	**3**

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	4	5	6	**7**
11	Haptic device force feedback.	1	2	3	4	5	**6**	7
12	Display of messages from the robot.	1	2	3	**4**	5	6	7
13	Display of measured forces.	1	**2**	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	5	**6**	7
15	Virtual display of active trajectories.	1	2	3	**4**	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	5	**6**	7
17	Transparency of vritual workpiece from robot confidence.	1	2	**3**	4	5	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files: test_s01_t211_rxxx — manual; _t213_rxxx — collaborative

Subject number: #1

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	-1	0	1	**2**	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	**1**	2	3
2a	I felt uncomfortable with the robot (manual mode).	**-3**	-2	-1	0	1	2	3
2b	I felt uncomfortable with the robot (collaboration mode).	**-3**	-2	-1	0	1	2	3
3a	The robot was trustworthy (manual mode).	-3	-2	-1	**0**	1	2	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	**0**	1	2	3
4a	The robot was intelligent (manual mode).	-3	-2	-1	**0**	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	**1**	2	3
5a	I found what I was doing with the robot confusing (manual mode).	**-3**	-2	-1	0	1	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	-3	**-2**	-1	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	1	2	**3**
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	2	**3**
8	I prefered manual mode over collaboration mode.	**-3**	-2	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	-3	-2	-1	0	1	2	**3**

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	4	5	6	**7**
11	Haptic device force feedback.	1	2	3	**4**	5	6	7
12	Display of messages from the robot.	1	2	**3**	4	5	6	7
13	Display of measured forces.	**1**	2	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	**5**	6	7
15	Virtual display of active trajectories.	1	2	3	**4**	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	5	**6**	7
17	Transparency of vritual workpiece from robot confidence.	1	**2**	3	4	5	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files	Subject number
s02_t211_r0-r04 - manual 213_r0-4 - collaborative	#2

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	-1	0	1	**2**	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	1	2	**3**
2a	I felt uncomfortable with the robot (manual mode).	-3	-2	-1	0	**1**	2	3
2b	I felt uncomfortable with the robot (collaboration mode).	-3	-2	-1	0	1	2	**3**
3a	The robot was trustworthy (manual mode).	-3	-2	-1	0	1	**2**	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	0	1	2	**3**
4a	The robot was intelligent (manual mode).	-3	-2	-1	**0**	1	X	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	1	2	**3**
5a	I found what I was doing with the robot confusing (manual mode).	-3	-2	-1	0	**1**	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	-3	**-2**	-1	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	3	-2	-1	0	1	2	**3**
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	2	**3**
8	I prefered manual mode over collaboration mode.	**-3**	-2	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	-3	-2	-1	0	1	2	**3**

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	4	**5**	6	7
11	Haptic device force feedback.	1	2	3	4	5	6	**7**
12	Display of messages from the robot.	1	2	3	**4**	5	6	7
13	Display of measured forces.	1	2	**3**	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	**5**	6	7
15	Virtual display of active trajectories.	1	**2**	3	4	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	**5**	6	7
17	Transparency of vritual workpiece from robot confidence.	1	2	3	**4**	5	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files	Subject number
manual TID211 collaborative TID213	#3

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	-1	0	1	**2**	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	**1**	2	3
2a	I felt uncomfortable with the robot (manual mode).	-3	**-2**	-1	0	1	2	3
2b	I felt uncomfortable with the robot (collaboration mode).	-3	-2	-1	**0**	1	2	3
3a	The robot was trustworthy (manual mode).	-3	-2	-1	0	1	**2**	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	0	1	**2**	3
4a	The robot was intelligent (manual mode).	-3	-2	-1	**0**	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	**1**	2	3
5a	I found what I was doing with the robot confusing (manual mode).	-3	**-2**	-1	0	1	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	-3	-2	**-1**	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	1	2	**3**
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	2	**3**
8	I prefered manual mode over collaboration mode.	-3	**-2**	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	-3	-2	-1	0	1	2	**3**

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	**4**	5	6	7
11	Haptic device force feedback.	1	2	3	4	5	**6**	7
12	Display of messages from the robot.	1	2	3	**4**	5	6	7
13	Display of measured forces.	**1**	2	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	5	6	**7**
15	Virtual display of active trajectories.	1	**2**	3	4	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	5	**6**	7
17	Transparency of vritual workpiece from robot confidence.	1	2	**3**	4	5	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	Top down view change was good. Needs dynamics comp.

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files		Subject number
manual TID 211 collaborative TID 213	11 X 1 X 1 1 X 1 1 1 1	# 4

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	-1	0	1	0	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	1	2	0
2a	I felt uncomfortable with the robot (manual mode).	0	-2	-1	0	1	2	3
2b	I felt uncomfortable with the robot (collaboration mode).	0	-2	-1	0	1	2	3
3a	The robot was trustworthy (manual mode).	0	-2	-1	0	1	2	3
3b	The robot was trustworthy (collaboration mode).	0	-2	-1	0	1	2	3
4a	The robot was intelligent (manual mode).	-3	-2	-1	0	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	1	0	3
5a	I found what I was doing with the robot confusing (manual mode).	0	-2	-1	0	1	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	0	-2	-1	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	1	2	0
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	2	0
8	I prefered manual mode over collaboration mode.	0	-2	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	0	-2	-1	0	1	2	3

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	0	5	6	7
11	Haptic device force feedback.	1	2	3	4	5	0	7
12	Display of messages from the robot.	1	2	0	4	5	6	7
13	Display of measured forces.	0	2	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	5	6	0
15	Virtual display of active trajectories.	0	2	3	4	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	5	6	0
17	Transparency of vritual workpiece from robot confidence.	1	2	3	4	0	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files	Subject number
TID211 s05 TID213 s05	#5

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	**-1**	0	1	2	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	**1**	2	3
2a	I felt uncomfortable with the robot (manual mode).	-3	-2	-1	0	1	**2**	3
2b	I felt uncomfortable with the robot (collaboration mode).	-3	-2	**-1**	0	1	2	3
3a	The robot was trustworthy (manual mode).	-3	-2	**-1**	0	1	2	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	0	**1**	2	3
4a	The robot was intelligent (manual mode).	-3	-2	**-1**	0	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	**1**	2	3
5a	I found what I was doing with the robot confusing (manual mode).	-3	-2	**-1**	0	1	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	-3	-2	**-1**	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	**1**	2	3
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	**2**	3
8	I prefered manual mode over collaboration mode.	-3	**-2**	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	-3	-2	-1	0	1	**2**	3

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	4	5	**6**	7
11	Haptic device force feedback.	1	2	**3**	4	5	6	7
12	Display of messages from the robot.	**1**	2	3	4	5	6	7
13	Display of measured forces.	**1**	2	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	**5**	6	7
15	Virtual display of active trajectories.	1	**2**	3	4	5	6	7
16	Virtual display of active hole selection.	1	2	3	**4**	5	6	7
17	Transparency of virtual workpiece from robot confidence.	1	2	3	4	**5**	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	Comparison to true manual comfort and mental load absent from questionnaire

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files	Subject number
TID211 HHT TID213 HHT	#6

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	-2	-1	0	1	6	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	1	2	6
2a	I felt uncomfortable with the robot (manual mode).	-3	-2	-1	0	1	2	3
2b	I felt uncomfortable with the robot (collaboration mode).	-3	-2	-1	0	1	2	3
3a	The robot was trustworthy (manual mode).	-3	-2	-1	0	1	2	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	0	1	2	3
4a	The robot was intelligent (manual mode).	-3	-2	-1	0	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	1	2	3
5a	I found what I was doing with the robot confusing (manual mode).	-3	-2	-1	0	1	2	3
5b	I found what I was doing with the robot confusing (collaboration mode)	-3	-2	-1	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	1	2	3
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	2	3
8	I prefered manual mode over collaboration mode.	-3	-2	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	-3	-2	-1	0	1	2	3

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	3	4	5	6	7
11	Haptic device force feedback.	1	2	3	4	5	6	7
12	Display of messages from the robot.	1	2	3	4	5	6	7
13	Display of measured forces.	1	2	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	5	6	7
15	Virtual display of active trajectories.	1	2	3	4	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	5	6	7
17	Transparency of virtual workpiece from robot confidence.	1	2	3	4	5	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	

User Questionnaire

Shared Autonomy for Manufacturing Telerobotics

Corresponding test files	Subject number
TID 211 ~~HHH~~ TID 213 ~~HHH~~	#7

Question	Please indicate to what extent you agree or disagree with each of the following statements.	Strongly disagree						Strongly agree
1a	The robot and I worked fluently together (manual mode).	-3	(-2)	-1	0	1	2	3
1b	The robot and I worked fluently together (collaboration mode).	-3	-2	-1	0	(1)	2	3
2a	I felt uncomfortable with the robot (manual mode).	-3	-2	-1	0	1	2	(3)
2b	I felt uncomfortable with the robot (collaboration mode).	-3	-2	-1	0	1	(2)	3
3a	The robot was trustworthy (manual mode).	-3	-2	-1	0	1	(2)	3
3b	The robot was trustworthy (collaboration mode).	-3	-2	-1	0	1	(2)	3
4a	The robot was intelligent (manual mode).	-3	(-2)	-1	0	1	2	3
4b	The robot was intelligent (collaboration mode).	-3	-2	-1	0	1	(2)	3
5a	I found what I was doing with the robot confusing (manual mode).	-3	-2	-1	0	1	2	(3)
5b	I found what I was doing with the robot confusing (collaboration mode)	-3	(-2)	-1	0	1	2	3
6	Collaboration mode was mentally easier than manual mode.	-3	-2	-1	0	1	(2)	3
7	Collaboration mode was physically easier than manual mode.	-3	-2	-1	0	1	(2)	3
8	I prefered manual mode over collaboration mode.	-3	(-2)	-1	0	1	2	3
9	Collaboration mode was faster than manual mode.	(-3)	-2	-1	0	1	2	(3)

Question	Please indicate to what extent you found the following features in the user workstation and interface useful to the task.	Least useful						Most useful
10	Display of live camera feed.	1	2	(3)	4	5	6	7
11	Haptic device force feedback.	1	(2)	3	4	5	6	7
12	Display of messages from the robot.	1	2	(3)	4	5	6	7
13	Display of measured forces.	(1)	2	3	4	5	6	7
14	Display of virtual robot in workspace.	1	2	3	4	(5)	6	7
15	Virtual display of active trajectories.	(1)	2	3	4	5	6	7
16	Virtual display of active hole selection.	1	2	3	4	5	(6)	7
17	Transparency of virtual workpiece from robot confidence.	1	(2)	3	4	5	6	7

Question	Please provide any additional comments or observations that you think might be helpful to the analysis of the experiments.
18	

Appendix C

SOFTWARE IMPLEMENTATIONS

Overview

This project implements the hole cleaning experiment for the 3DOF HEBI actuator robot and teleoperation interface. MATLAB code implements learning algorithms, rapid data collection, and data post processing. C++ code implements the runtime experiment. The folder is organized according to the following.

- `matlab/` - this folder contains all the matlab code for learning, data processing and any runtime tests
- `resources/` - this folder contains robot geometry models
- `src/` - this folder contains the multithreaded runtime source code, including the main functions and supporting control loop threads
- `lib/` - this folder contains class definitions to implement objects needed for shared autonomy
- `include/` - this folder contains the autogenerated constants header file from `matlab/knet.m`
- `tests/` - this folder contains unit tests for the classes in `lib`

MATLAB Code

These files were tested with MATLAB 2017a. Note that both GMM-GMR-v2.0 and the HEBI API need to be on the MATLAB path for these classes. It's easiest to just copy them into the `matlab/` folder. GMM-GMR-v2.0 can be downloaded from https://www.mathworks.com/matlabcentral/fileexchange/19630, and the HEBI API

(v1.5.1) can be downloaded from the http://docs.hebi.us/tools.html#matlab-api.

- `matlab/knet.m` - (class) the policy learning class that implements the learning algorithm methods
- `matlab/learnTask.m` - a sample script to learn the brush cleaning task data using knet
- `matlab/test_taskHardware.m` - tests the learned policy on the robot hardware
- `matlab/test_demonstration.m` - records training data on the robot hardware
- `matlab/localizeTargetOnline.m` - for target localization during the compiled C++ experiment run (see runtime note below)
- `matlab/postProcessData.m` - (function) for post-processing the data files
- `matlab/trajGenerator.m` - (function) minimum jerk trajectory generator
- `matlab/loadDynamics.m` - (function) load the manipulator dynamics matrices
- `matlab/loadKinematics.m` - (function) load the forward kinematics and jacobian anonymous functions
- `matlab/resample.m` - (function) particle systematic resampling (particle filter)

In addition to the MATLAB scripts above, the following files are written for mex compile to matlab in order to interface with hardware

- `matlab/atiConnection.cpp` - connects to the ATI Mini-45 F/T sensor. See file for compile instructions

- `matlab/hapticConnection.cpp` - connects to the Force Dimension Omega.7 haptic device. See file for compile instructions

Runtime Experiments

A CMakeLists file has been provided to generate project files for your platform and compiler of choice. It has been tested on Mac 10.12.6 using XCode 9.2, and Windows 8.1 Enterprise

using Visual Studio 2015 Community. Follow the installation procedure outlined below to install for your platform/compiler of choice. You may need to set your path (environmental variables) depending on how you install the following libraries - ultimately the CMake command `find_package` needs to know where to look for everything. For more about the dependencies, see CMakeLists.txt. Once the project has been built, the executables can be compiled into Release. If you use the CMake instructions below, this will create an executable in `build/Release/` called `app-brush`. If you have GTest installed (optional), a unit test suite named `app-tests` will also be compiled. To run the experiment, simply run `app-brush`.

Installation on Mac

1. Download, build and install Cmake (version 3.11.4). The homebrew install is `brew install cmake`.
2. Download, build and install (using CMake) OpenCV (version 3.4.0).
3. Download, build and install (using CMake) CoreRobotics (version 1.8.0).
4. Download, build and install (using CMake) Chai3d (version 3.1.1)
5. Download and build (using CMake) Hebi C++ API (version 1.0)
6. Create a subdirectory in the project root named `build/`, and build the project files in it (using CMake `cmake -G "Xcode" -DHEBI_DIR=<path to hebi root> ..` from within the build folder).

Installation on Windows (not recommended)

1. Download, build and install Cmake.
2. Download, build and install (using CMake) OpenCV
3. Add OpenCV DLLs to path: `PATH=%PATH%;<path to opencv dlls>`
4. Download CRexternal repository (contains Eigen and boost)
5. Download, build and install (using CMake) CoreRobotics
6. Download, build and install (using CMake) Chai3d

7. Build freeglut in Chai3d
8. Download and build (using CMake) hebi
9. Create a subdirectory in the project root named `build/`, and build the project files in it (using CMake `cmake -G "Visual Studio 14 2015" -DHEBI_DIR=<path to hebi root> -DCMAKE_PREFIX_PATH=<path to opencv build> ..` from within the build folder).

Additional steps for particle filter localization

If you would like to have particle filter localization active during the experiment, extra steps are needed since the runtime particle filter script is in MATLAB. You need to additionally install CoreRobotics for MATLAB, i.e. set -Dmatlab=true during CMake install. Note this was only tested on Mac, extra steps may need to be taken to get this working on Windows. Once the CoreRobotics MATLAB is built, CoreRobotics functions are available in MATLAB. Set the CoreRobotics path in `matlab/localizeTargetOnline.m` to your installation of the CoreRobotics MATLAB library. To include localization, run the scripts in the following order 1. Run the compiled executable `build/Release/app-brush` first (it acts as the shared memory server) 2. Then run `matlab/localizeTargetOnline.m` (it acts as the shared memory client and will only run if the server is active). This will run in the background and receive instructions from app-brush during testing. Note that you will need to manually exit the matlab script with `ctrl-c` when you are done, since it will continue to run forver.

C.1 Learning algorithm (MATLAB)

The following MATLAB code implements the learning algorithm and related routines for data collection and processing.

C.1.1 Learning class

Listing C.1: matlab/knet.m - class for learning policies from demonstration data

```
% Copyright 2014-2019, Parker Owan
% PhD Dissertation, University of Washington
classdef knet < handle
    % knet "KineticNet"
    %
    % Summary:
    % This class implements imitation learning of kinetic manufacturing
    % tasks from task data. Note that this code implements the methods
    % discussed in Chapter 5 of the thesis.

    % Public properties
    properties
        mMeasurements  % measurement noise models
        mCenter        % model centroid offset (position)
        mGmm           % cluster model
        mStates        % model states
        mActions       % model actions
        mTransition    % NARX transition model (NN)
        mTrajectories  % trajectories of individual actions
        mChannels      % channels of the associated objects
        mInternalStates % internal states
    end


    % Primary methods
    methods

        function obj = knet()
```

```
29          % Summary of constructor
30          %
31          % this constructor adds the GMM-GMR-v2.0 package to the matlab
32          % path. this package can be downloaded from matlab file
33          % exhange at:
34          % https://www.mathworks.com/matlabcentral/fileexchange/
35          % 19630-gaussian-mixture-model-gmm-
36          % gaussian-mixture-regression-gmr
37          addpath GMM-GMR-v2.0
38      end
39
40      function populate(obj, model, traj, ch, center, noise)
41          % populate(obj, model, traj, ch, center, noise)
42          %
43          % Create a knet object with populated data member variables
44          %
45          % obj     - the knet object
46          % noise   - the data structure (with measurement noise
47          % model   - the learned model (i.e. selected from training)
48          % traj    - the learned action trajectories
49          % ch      - the channel structure used for training
50          % center  - the data centroid
51          % noise   - the measurement noise model
52
53          % push the structures to the appropriate model member vars
54          obj.mMeasurements = noise;
55          obj.mCenter = center;
56          obj.mGmm = model.GMM;
57          obj.mStates = model.States;
```

```
58          obj.mActions = model.Actions;
59          obj.mTransition = model.Transition;
60          obj.mTrajectories = traj;
61          obj.mChannels = ch;
62      end
63
64      function setup(obj)
65          % setup(obj)
66          %
67          % Sets up common internal states for knet use. Further init
68          % calls are needed for specific operations (e.g. for
69          % simulation, runtime policy, etc...)
70          %
71          % obj     - the knet object
72
73          % add GMR to path
74          addpath GMM-GMR-v2.0
75
76          % generate the transition function
77          genFunction(obj.mTransition.net,...
78              obj.mTransition.fcnName,...
79              'MatrixOnly','yes',...
80              'ShowLinks','no');
81
82          % find the start action
83          fa = obj.mTransition.FirstAction;
84          idx0 = find(fa == max(fa),1,'first');
85          obj.mInternalStates.FirstAction = idx0;
86
```

```
87          % find the stop action
88          fa = obj.mTransition.LastAction;
89          idx1 = find(fa == max(fa),1,'first');
90          obj.mInternalStates.LastAction = idx1;
91      end
92
93      function writeParameterFile(obj)
94          % writeParameterFile(obj)
95          %
96          % The function autocodes the model parameters to a parameter
97          % header file to be used by the C++ experiment code.
98          %
99          % obj     - the knet object
100
101         % get some dimensions
102         [d, ¬] = size(obj.mTrajectories(1).x);
103         na = numel(obj.mTrajectories);
104         nk = obj.mGmm.NumComponents;
105
106         % get the first and last actions
107         A0 = find(obj.mTransition.FirstAction ==...
108             max(obj.mTransition.FirstAction), 1, 'first');
109         Af = find(obj.mTransition.LastAction ==...
110             max(obj.mTransition.LastAction), 1, 'first');
111
112         % create the parameter header file
113         fid = fopen('../include/KnetConstants.h','w');
114         fprintf(fid, '// Parameter file for Knet.h\n');
115         fprintf(fid, ['// Autogenerated by knet.m ',...
```

```
116             '(method: writeParameterFile)\n']);
117         fprintf(fid, '// Copyright (2018) Parker Owan\n\n\n');
118         fprintf(fid, '#ifndef KNET_CONSTANTS_H_\n');
119         fprintf(fid, '#define KNET_CONSTANTS_H_\n\n');
120         fprintf(fid, '#include "CoreRobotics.hpp"\n');
121         fprintf(fid, '#include "Trajectory.h"\n');
122         fprintf(fid, '#include "Knet.h"\n\n');
123         fprintf(fid, 'using namespace cr;\n');
124         fprintf(fid, 'using namespace cr::control;\n\n');
125         fprintf(fid, '// DIMENSIONS\n');
126         fprintf(fid, '#define KNET_DIM      \t%i\n',d);
127         fprintf(fid, '#define KNET_ACTIONS  \t%i\n',na);
128         fprintf(fid, '#define KNET_MIXTURES \t%i\n',nk);
129         fprintf(fid, '#define KNET_FIRST_ACTION \t%i\n',A0-1);
130         fprintf(fid, '#define KNET_LAST_ACTION \t%i\n\n',Af-1);
131
132         % write trajectories load
133         fprintf(fid, '// TRAJECTORIES\n');
134         fprintf(fid, ['void loadKnetTrajectories(Trajectory ',...
135             '(&traj)[KNET_ACTIONS]) {\n\n']);
136         fprintf(fid, '\tWaypoint wp;\n');
137         fprintf(fid, '\tEigen::VectorXd x(KNET_DIM);\n');
138         fprintf(fid, '\tEigen::VectorXd v(KNET_DIM);\n\n');
139         for k = 1:na
140             ns=length(obj.mTrajectories(k).t);
141             for p = 1:ns
142                 fprintf(fid, '\t// Trajectory %i, Point %i\n',k-1,p-1);
143                 fprintf(fid, '\tx << %+.10f, %+.10f;\n',...
144                     obj.mTrajectories(k).x(:,p));
```

```
145                 fprintf(fid, '\tv << %+.10f, %+.10f;\n',...
146                     obj.mTrajectories(k).v(:,p));
147                 fprintf(fid, '\twp.time = %+.10f;\n',...
148                     obj.mTrajectories(k).t(p));
149                 fprintf(fid, '\twp.position = x;\n');
150                 fprintf(fid, '\twp.velocity = v;\n');
151                 fprintf(fid, '\ttraj[%i].add(wp);\n\n',k-1);
152             end
153         end
154         fprintf(fid, '}\n\n');
155 
156         % write mixture model load
157         fprintf(fid, '// MIXTURE MODELS\n');
158         fprintf(fid, 'void loadMixtureModels(Gmm (&gmm)[KNET_ACTIONS]) {');
159         fprintf(fid, '\n\n');
160         fprintf(fid, '\tNoiseGaussian gaussian[KNET_MIXTURES];\n');
161         fprintf(fid, '\tEigen::VectorXd Mu(4 * KNET_DIM);\n');
162         fprintf(fid, ['\tEigen::MatrixXd Sigma(4 * ',...
163             'KNET_DIM, 4 * KNET_DIM);']);
164         fprintf(fid, '\n\n');
165         for k = 1:nk
166             Mu = obj.mGmm.mu(k,:);
167             Sigma = obj.mGmm.Sigma(:,:,k);
168             fprintf(fid, '\t// Gaussian Cluster %i\n',k-1);
169             fprintf(fid, '\tMu << ');
170             for i = 1:length(Mu)-1
171                 fprintf(fid, '%+.10f, ',Mu(i));
172             end
173             fprintf(fid, '%+.10f;\n',Mu(end));
```

```
174                 fprintf(fid, '\tSigma << ');
175                 for i = 1:length(Mu)
176                     for j = 1:length(Mu)-1
177                         fprintf(fid, '%+.10f, ',Sigma(i,j));
178                     end
179                     if i == length(Mu)
180                         fprintf(fid, '%+.10f;\n',Sigma(i,end));
181                     else
182                         fprintf(fid, '%+.10f,\n',Sigma(i,end));
183                     end
184                 end
185                 fprintf(fid, '\tgaussian[%i].setParameters(Sigma, Mu);',k-1);
186                 fprintf(fid, '\n\n');
187             end
188             for a = 1:na
189                 fprintf(fid, '\t// Action manifold %i\n',a-1);
190                 A = obj.mActions{a};
191                 for i = 1:length(A)
192                     weight = obj.mGmm.ComponentProportion(A(i));
193                     fprintf(fid, '\tgmm[%i].add(gaussian[%i], %+.10f);\n',...
194                         a-1, A(i)-1, weight);
195                 end
196                 fprintf(fid, '\n');
197             end
198             fprintf(fid, '}\n\n');
199
200             % write neural network model load
201             fprintf(fid, '// NEURAL NETWORK PARAMETERS\n');
202             fprintf(fid, ['void loadNeuralParameters(NeuralParameters ',...
```

```
203             '&nnet) {\n\n']);
204         nnet = obj.mTransition.net;
205
206         [r, c] = size(nnet.IW{1});
207         fprintf(fid, '\tEigen::MatrixXd IW1_1(%i,%i);\n',r,c);
208         fprintf(fid, '\tIW1_1 << \n');
209         for i = 1:r
210             for j = 1:c-1
211                 fprintf(fid, '%+.10f, ',nnet.IW{1}(i,j));
212             end
213             if i == r
214                 fprintf(fid, '%+.10f;\n\n',nnet.IW{1}(i,end));
215             else
216                 fprintf(fid, '%+.10f,\n',nnet.IW{1}(i,end));
217             end
218         end
219
220         [r, c] = size(nnet.LW{2,1});
221         fprintf(fid, '\tEigen::MatrixXd LW2_1(%i,%i);\n',r,c);
222         fprintf(fid, '\tLW2_1 << \n');
223         for i = 1:r
224             for j = 1:c-1
225                 fprintf(fid, '%+.10f, ',nnet.LW{2,1}(i,j));
226             end
227             if i == r
228                 fprintf(fid, '%+.10f;\n\n',nnet.LW{2,1}(i,end));
229             else
230                 fprintf(fid, '%+.10f,\n',nnet.LW{2,1}(i,end));
231             end
```

```
232             end
233
234             [r, c] = size(nnet.b{1});
235             fprintf(fid, '\tEigen::VectorXd b1(%i);\n',r);
236             fprintf(fid, '\tb1 << \n');
237             for i = 1:r-1
238                 fprintf(fid, '%+.10f, ',nnet.b{1}(i));
239             end
240             fprintf(fid, '%+.10f;\n\n',nnet.b{1}(end));
241
242             [r, c] = size(nnet.b{2});
243             fprintf(fid, '\tEigen::VectorXd b2(%i);\n',r);
244             fprintf(fid, '\tb2 << \n');
245             for i = 1:r-1
246                 fprintf(fid, '%+.10f, ',nnet.b{2}(i));
247             end
248             fprintf(fid, '%+.10f;\n\n',nnet.b{2}(end));
249
250             settings = nnet.inputs{1}.processSettings{1};
251             [r, c] = size(settings.gain);
252             fprintf(fid, '\tEigen::VectorXd gain(%i);\n',r);
253             fprintf(fid, '\tgain << \n');
254             for i = 1:r-1
255                 fprintf(fid, '%+.10f, ',settings.gain(i));
256             end
257             fprintf(fid, '%+.10f;\n\n',settings.gain(end));
258
259             [r, c] = size(settings.xoffset);
260             fprintf(fid, '\tEigen::VectorXd xoffset(%i);\n',r);
```

```
261             fprintf(fid, '\txoffset << \n');
262             for i = 1:r-1
263                 fprintf(fid, '%+.10f, ',settings.xoffset(i));
264             end
265             fprintf(fid, '%+.10f;\n\n',settings.xoffset(end));
266
267             fprintf(fid, '\tdouble ymin = %+.10f;\n\n',settings.ymin);
268
269             fprintf(fid, '\t// Write to the nnet parameter struct\n');
270             fprintf(fid, '\tnnet.il_ymin = ymin;\n');
271             fprintf(fid, '\tnnet.il_xoffset = xoffset;\n');
272             fprintf(fid, '\tnnet.il_gain = gain;\n');
273             fprintf(fid, '\tnnet.l1_b    = b1;\n');
274             fprintf(fid, '\tnnet.l1_W    = IW1_1;\n');
275             fprintf(fid, '\tnnet.l2_b    = b2;\n');
276             fprintf(fid, '\tnnet.l2_W    = LW2_1;\n\n');
277             fprintf(fid, '}\n\n');
278
279             % end the header
280             fprintf(fid, '#endif\n');
281             fclose(fid);
282         end
283
284         function policyInit(obj, t, w)
285             % policyInit(obj, t, w)
286             %
287             % Initializes the internal states at time t with work w.
288             %
289             % obj     - the knet object
```

```
290         % t       - current time (for the trajectory)
291         % w       - the work accumulated since the policy start
292
293         % get the first action
294         idx = obj.mInternalStates.FirstAction;
295
296         % set up the current action index
297         obj.mInternalStates.ActionIndex = idx;
298
299         % set up the work
300         obj.mInternalStates.Work = w;
301
302         % set up the runtime flags
303         obj.mInternalStates.Done = false;
304         obj.mInternalStates.T0 = t;
305         obj.mInternalStates.Tmax = obj.mTrajectories(idx).t(end);
306     end
307
308     function [x, v, a, f] = policyStep(obj, t, w)
309         % [x, v, a, f] = policyStep(obj, t, w)
310         %
311         % Step the policy given current time t and acucmulated work w
312         % since the beginning of the policy.
313         %
314         % obj     - the knet object
315         % t       - current time (for the trajectory)
316         % w       - the work accumulated since the policy start
317         % x       - position
318         % v       - velocity
```

```
319             % a       - acceleration
320             % f       - effort
321
322             % get the channels
323             ch = obj.mChannels;
324
325             % if we've surpased the max runtime on the trajectory, advance
326             % the current action index
327             if ((t - obj.mInternalStates.T0) ≥obj.mInternalStates.Tmax)
328                 obj.mInternalStates.Done = obj.policyAdvance(w);
329                 obj.mInternalStates.Tmax = ...
330                     obj.mTrajectories(obj.mInternalStates.ActionIndex).t(end);
331                 obj.mInternalStates.T0 = t;
332             end
333
334             if (¬obj.mInternalStates.Done)
335
336                 % if not done, step the trajectory
337                 [x,v] = obj.policyTraj(t-obj.mInternalStates.T0);
338             else
339
340                 % if done, fix the time to the end time
341                 t = obj.mTrajectories(obj.mInternalStates.LastAction).t(end);
342                 [x,v] = obj.policyTraj(t);
343             end
344
345             % predict the forces and accelerations from the trajectory
346             action = obj.mActions{obj.mInternalStates.ActionIndex};
347             a = knet.predictAccel(t, [x v], action, obj.mGmm, obj.mChannels);
```

```
348             f = knet.predictForce(t, [x v], action, obj.mGmm, obj.mChannels);
349         end
350 
351         function done = policyAdvance(obj, w)
352             % done = policyAdvance(obj, w)
353             %
354             % Advance the policy and check if done
355             %
356             % obj     - the knet object
357             % w       - the work accumulated since the policy start
358             % done    - flag indicating if the policy is done
359 
360             % set the flag
361             done = false;
362 
363             % get the current action index
364             idx =obj.mInternalStates.ActionIndex;
365 
366             % check if this is the last action
367             if (idx == obj.mInternalStates.LastAction)
368 
369                 % if so, then we're done
370                 done = true;
371             else
372 
373                 % if not, then advance to the next action
374                 Pa = zeros(1,length(obj.mActions));
375                 Pa(idx) = 1;
376                 Wa = knet.actionLikelihood(Pa, w, obj.mTransition);
```

```
377                 obj.mInternalStates.ActionIndex =...
378                     find(Wa == max(Wa),1,'first');
379             end
380
381         end
382
383         function [x, v] = policyTraj(obj, t)
384             % [x, v] = policyTraj(obj, t)
385             %
386             % Generate the motion(position and velocity) for the current
387             % action index from time t.
388             %
389             % obj     - the knet object
390             % t       - current time (for the trajectory)
391             % x       - trajectory position
392             % v       - trajectory velocity
393
394             % return the trajectory
395             traj = obj.mTrajectories(obj.mInternalStates.ActionIndex);
396
397             % interpolate the trajectory for the current time
398             x = interp1(traj.t', traj.x', t');
399             v = interp1(traj.t', traj.v', t');
400         end
401
402     end
403
404
405     % Predictors, probability, likelihood, and training
```

```
406     methods (Static)
407
408         function P = cluster(gmm, D, vars)
409             % P = cluster(gmm, D, vars)
410             %
411             % Clusters the data in D with gmm along indices in 'vars'
412             %
413             % gmm     - the gaussian mixture model
414             % D       - the data to evaluate (observations x channels)
415             % vars    - the corresponding columns in D for gmm dimensions
416             % P       - posterior cluster density
417             P = zeros(size(D,1),gmm.NumComponents);
418             for i = 1:gmm.NumComponents
419                 if strcmp(gmm.CovarianceType,'diagonal')
420                     Sigma = diag(gmm.Sigma(1,vars,i));
421                 else
422                     Sigma = gmm.Sigma(vars,vars,i);
423                 end
424                 P(:,i) = mvnpdf(D,...
425                     gmm.mu(i,vars), Sigma .*...
426                     gmm.ComponentProportion(i));
427             end
428             P = P ./ sum(P,2);
429         end
430
431         function [f, fcov] = predictForce(t, xv, manifold, gmm, ch)
432             % [f,fcov] = predictForce(t, xv, action, gmm, ch)
433             %
434             % Predicts the task force f from state x.
```

```
435             %
436             % t       - time
437             % xv      - [observations x (position, velocity)]
438             % manifold - indices of the gmm to be used for prediction
439             % gmm     - the gaussian mixture model
440             % ch      - channel structure
441             % f       - mean - [observations x (force)]
442             % fcov    - covariance - [(force) x (force) x observations]
443             [f,fcov] = GMR(gmm.ComponentProportion(manifold),...
444                 gmm.mu(manifold,:)',...
445                 gmm.Sigma(:,:,manifold),...
446                 xv',[ch.x ch.v],ch.f);
447             f = f';
448         end
449
450         function [a, acov] = predictAccel(t, xv, manifold, gmm, ch)
451             % [a,acov] = predictAccel(t, xv, action, gmm, ch)
452             %
453             % Predicts the acceleration a from state x.
454             %
455             % t       - time
456             % xv      - [observations x (position, velocity)]
457             % manifold - indices of the gmm to be used for prediction
458             % gmm     - the gaussian mixture model
459             % ch      - channel structure
460             % a       - [observations x (accel)]
461             % acov    - covariance - [(accel) x (accel) x observations]
462             [a,acov] = GMR(gmm.ComponentProportion(manifold),...
463                 gmm.mu(manifold,:)',...
```

```
464             gmm.Sigma(:,:,manifold),...
465             xv',[ch.x ch.v],ch.a);
466         a = a';
467     end
468
469     function [xdot, acov] = predictMotion(t, xv, manifold, gmm, ch, sample)
470         % [xdot, acov] = predictMotion(t, xv, manifold, gmm, ch, sample)
471         %
472         % Predicts the state derivative xdot from state x.
473         %
474         % t       - time
475         % x       - [observations x (position, velocity)]
476         % manifold - indices of the gmm to be used for prediction
477         % gmm     - the gaussian mixture model
478         % ch      - channel structure
479         % sample  - set high to sample from the noise model
480         % xdot    - [observations x (velocity, acceleration)]
481         % acov    - [accel x accel x observations]
482         if nargin < 6
483             sample = 0;
484         end
485         [n,d] = size(xv);
486         d = d/2;
487         [a_mu, a_cov] = GMR(gmm.ComponentProportion(manifold),...
488             gmm.mu(manifold,:)',...
489             gmm.Sigma(:,:,manifold),...
490             xv',[ch.x ch.v],ch.a);
491         a_mu = a_mu';
492         if sample > 0
```

```
493             s = randn(n,d);
494             w = zeros(size(s));
495             for i = 1:d
496                 w(:,i) = sample .* s(:,i) .* sqrt(squeeze(a_cov(i,i,:)));
497             end
498             a_mu = a_mu + w;
499         end
500         xdot = [xv(:,d+1:end), a_mu];
501         acov = a_cov;
502     end
503 
504     function L = regrLikelihood(D, manifold, gmm, ch)
505         % L = regrLikelihood(D, manifold, gmm, ch)
506         %
507         % computes the regression likelihood on D for a given manifold
508         %
509         % D       - the data to evaluate (observations x channels)
510         % manifold - indices of the gmm to be used for prediction
511         % gmm     - the gaussian mixture model
512         % ch      - channel structure
513         % L       - computed likelihood (observations-1 x 1)
514 
515         % compute time step
516         Delta = D(2:end,ch.t)-D(1:end-1,ch.t);
517 
518         % intput/output
519         in = [ch.x ch.v];
520         out = [ch.a ch.f];
521 
```

```
522             % compute the regression likelihoods
523             likF = zeros(size(D,1)-1,1);
524             likA = zeros(size(D,1)-1,1);
525
526             % predict force and acceleration
527             [f_mu, f_cov] = knet.predictForce(0,...
528                 D(1:end-1,[ch.x ch.v]), manifold, gmm, ch);
529             [a_mu, a_cov] = knet.predictAccel(0,...
530                 D(1:end-1,[ch.x ch.v]), manifold, gmm, ch);
531
532             % integrate the acceleration (forward Euler)
533             v_mu  = Delta .* a_mu + D(1:end-1,ch.v);
534             v_cov = zeros(size(a_cov));
535             for i = 1:length(a_cov)
536                 v_cov(:,:,i) = Delta(i).^2 * a_cov(:,:,i);
537             end
538
539             % compute the likelihoods
540             try
541                 likF = mvnpdf(f_mu, D(1:end-1,ch.f), f_cov);
542                 likA = mvnpdf(a_mu, D(1:end-1,ch.a), a_cov);
543                 likV = mvnpdf(v_mu, D(2:end,ch.v), v_cov);
544             catch
545                 disp('Error evaluating likelihoods')
546                 for i = 1:size(D,1)-1
547                     fc = f_cov(:,:,i);
548                     ac = a_cov(:,:,i);
549                     vc = v_cov(:,:,i);
550                     fnan(i) = isnan(sum(sum(fc))); %check for nans
```

```
551                 anan(i) = isnan(sum(sum(ac))); %check for nans
552                 vnan(i) = isnan(sum(sum(vc))); %check for nans
553                 fNotSym(i) = sum(sum((fc - fc').^2)) > 0;
554                 aNotSym(i) = sum(sum((ac - ac').^2)) > 0;
555                 vNotSym(i) = sum(sum((vc - vc').^2)) > 0;
556                 fNotPDef(i) = min(eig(fc)) ≤0;
557                 aNotPDef(i) = min(eig(ac)) ≤0;
558                 vNotPDef(i) = min(eig(vc)) ≤0;
559
560                 % make symmetric
561                 f_cov(:,:,i) = fc - (fc - fc')/2;
562                 a_cov(:,:,i) = ac - (ac - ac')/2;
563                 v_cov(:,:,i) = vc - (vc - vc')/2;
564             end
565             nanCov = (sum(fnan)>0) | (sum(anan)>0) | (sum(vnan)>0);
566             asymCov = (sum(fNotSym)>0) | (sum(aNotSym)>0) |...
567                 (sum(vNotSym)>0);
568             npdefCov = (sum(fNotPDef)>0) | (sum(aNotPDef)>0)|...
569                 (sum(vNotPDef)>0);
570
571             %
572             if (asymCov && ((¬nanCov) && (¬npdefCov)))
573                 disp('Attempting symmtery fix')
574                 likF = mvnpdf(f_mu, D(1:end-1,ch.f), f_cov);
575                 likA = mvnpdf(a_mu, D(1:end-1,ch.a), a_cov);
576                 likV = mvnpdf(v_mu, D(2:end,ch.v), v_cov);
577             else
578                 disp('Matrix is not positive definite or nan')
579                 keyboard
```

```
580             end
581         end
582
583         % output
584         L = likF .* likV;
585     end
586
587     function [xs, is] = sample(gmm, manifold, ns)
588         % [xs, is] = sample(gmm, manifold, ns)
589         %
590         % Samples from the manifold
591         %
592         % gmm     - the gaussian mixture model
593         % manifold - indices of the gmm to be used for prediction
594         % ns      - number of samples
595         % xs      - sampled data (ns x gmm vars)
596         % is      - sampled cluster index (ns x gmm vars)
597         priors = gmm.ComponentProportion(manifold);
598         priors = priors / sum(priors);
599         cdist = [0, cumsum(priors)];
600         isamp = rand(ns,1);
601         xs = zeros(ns,gmm.NumVariables);
602         is = zeros(ns,1);
603         for i = 2:length(cdist)
604             id = find((cdist(i-1) ≤isamp) & (isamp < cdist(i)));
605             R = gmm.Sigma(:,:,manifold(i-1));
606             xs(id,:) = gmm.mu(manifold(i-1),:) +...
607                 randn(length(id),gmm.NumVariables)*chol(R);
608             is(id,:) = manifold(i-1);
```

```
609             end
610         end
611
612         function p = gmmPdf(D, gmm, vars)
613             % p = gmmPdf(D, gmm, ch)
614             %
615             % Compute the Gaussian Mixture Model (GMM) pdf for the data in
616             % D with variables in ch.
617             %
618             % D       - data matrix [observations x vars]
619             % gmm     - the gaussian mixture model
620             % vars    - indices of gmm that corresponds to cols in D
621             % p       - pdf of gmm [observations x 1]
622             p = zeros(size(D,1),1);
623             for i = 1:gmm.NumComponents
624                 p = p + gmm.ComponentProportion(i) *...
625                     mvnpdf(D, gmm.mu(i,vars), gmm.Sigma(vars,vars,i));
626             end
627         end
628
629         function Pt = transLikelihood(La, mahal, trans, A, S)
630             % Pt = transLikelihood(La, mahal, trans, A, S)
631             %
632             % Computes the likelihood of transitioning given list of action
633             % indices La and the mahalanobis distances to the target states
634             %
635             % La      - list of action indices [observations x 1]
636             % mahal   - mahal distance to states [observations x states]
637             % trans   - the transition struct (learned in learnSequence)
```

```
638             % A       - cell set of actions
639             % S       - list of states
640             % Pt      - probability of transitioning [observations x 1]
641             Pt = zeros(size(La));
642             for a = 1:length(A)
643                 s = S == A{a}(end);
644                 id = La == a;
645                 Pt(id) = 1-trans.ptrans(a).cdf(mahal(id,s));
646             end
647         end
648 
649         function Wa = actionLikelihood(Pa, W, trans)
650             % Wa = actionLikelihood(Pa, W, trans)
651             %
652             % This function predicts the probability of a new action (i.e.
653             % an adjacent action, given the current action.
654             %
655             % Pa      - prob of current action [observations x actions]
656             % W       - work done so far in the task [1 x 1]
657             % trans   - the transition struct (learned in learnSequence)
658             % Wa      - prob of adjacent action [observations x actions]
659             U = [W*ones(size(Pa,1),1), Pa]';
660             if ¬isempty(trans.fcnName)
661                 Wa = feval(trans.fcnName, U)';
662             else
663                 Wa = trans.net(U)';
664             end
665         end
666 
```

```
667 function [Pa, Ptrans] = predictAction(Pa, W, mahal, trans, A, S)
668     % [Pa, Ptrans] = predictAction(Pa, W, mahal, trans, A, S)
669     %
670     % This function predicts the probability of the next action
671     % given the current action, accumulated work, mahalanobis
672     % distances to the states.
673     %
674     % Pa      - prob of current action [observations x actions]
675     % W       - work done so far in the task [1 x 1]
676     % mahal   - mahal distance to states [observations x ns]
677     % trans   - the transition struct (learned in learnSequence)
678     % A       - actions
679     % S       - states
680     % Pa      - prob of next action [observations x actions]
681     % Ptrans  - prob of trans to action [observations x actions]
682
683     % for each action, find the probability of transitioning
684     Ptrans = zeros(size(Pa));
685     for a = 1:length(A)
686         s = S == A{a}(end);
687         Ptrans(:,a) = 1-trans.ptrans(a).cdf(mahal(:,s));
688     end
689
690     % probability of transitioning/staying
691     Pt = diag(Ptrans);
692     Ps = diag(1-Ptrans);
693
694     % construct the diagonal matrix (i.e. stay in same state)
695     Da = eye(length(A));
```

```
696
697             % get the adjacency matrix (neural network
698             U = [W*ones(1,length(A)); Da]; % nnet input
699             if ¬isempty(trans.fcnName)
700                 Wa = feval(trans.fcnName, U);
701             else
702                 Wa = trans.net(U);
703             end
704
705             % compute the probability of next state
706             Pa = ((Ps * Da + Pt * Wa) * Pa')';
707
708             % normalize
709             Pa = Pa ./ sum(Pa,2);
710         end
711
712         function [logLik, ns] = likelihood(data, gmm, trans, A, S, useSeq)
713             % [logLik, ns] = likelihood(data, gmm, trans, A, S, useSeq)
714             %
715             % Compute the data likelihood.
716             %
717             % data     - the populated data structure
718             % gmm      - the mixture model
719             % trans    - the transition struct
720             % A        - cell set of actions
721             % S        - list of states
722             % useSeq   - flag (true = use forward chaining actions)
723             % logLik   - the log likelihood of the data
724             % ns       - number of samples used to compute the likelihood
```

```
725
726             % for each run in the test set
727             ns = zeros(1,length(data));
728             logLik = zeros(1,length(data));
729             for k = 1:length(data)
730                 [D, ch] = knet.processData(data(k));
731                 id = data(k).id_cluster;
732                 id = sort(id);
733                 D = D(id,:);
734                 Delta = [0; D(2:end,ch.t)-D(1:end-1,ch.t)];
735                 Work = cumsum(dot(-D(:,ch.f),D(:,ch.v),2).*Delta);
736
737                 % intput/output
738                 in = [ch.x ch.v];
739                 out = [ch.a ch.f];
740
741                 % compute mahal to state
742                 stateMahal = zeros(size(D,1),length(S));
743                 for s = 1:length(S)
744                     [¬,stateMahal(:,s)] = knet.gaussian(D(:,in)',...
745                         gmm.mu(S(s),in)',gmm.Sigma(in,in,S(s)));
746                 end
747
748                 % for each sample, forward chain the action probability
749                 La = zeros(size(D,1),length(A)); % action probability
750                 La(1,:) = trans.FirstAction; % probability
751                 Pt = zeros(size(D,1),length(A));
752                 for i = 2:size(D,1)
753
```

```
754                 % roll out the action prediction
755                 [La(i,:), Pt(i,:)] = knet.predictAction(...
756                     La(i-1,:), Work(i-1,:), stateMahal(i-1,:),...
757                     trans, A, S);
758             end
759
760             % compute the regression likelihoods
761             rLik = zeros(size(D,1)-1,length(A));
762             for a = 1:length(A)
763                 rLik(:,a) = knet.regrLikelihood(D, A{a}, gmm, ch);
764             end
765
766             % sum the data into the log likelihood
767             if (useSeq)
768                 llik = sum(log(sum(rLik .* La(1:end-1,:),2)));
769             else
770                 llik = sum(log(sum(rLik,2)));
771             end
772             logLik(k) = llik;
773             ns(k) = size(rLik,1);
774         end
775     end
776
777     function [A, B, C, D, Q, R] = stateSpace(gmm, ch)
778         % [A, B, C, D, Q, R] = stateSpace(gmm, ch)
779         %
780         % \dot{\chi} = A \chi + B + \nu
781         % f = C \chi + D + \epsilon
782         %
```

```
783         % where:
784         % \nu \sim \mathcal{N}(0,Q)
785         % \epsilon \sim \mathcal{N}(0,R)
786         %
787         % Compute the state space representation of the cluster model
788         % using Gaussian mixture regression. Note that this returns
789         % the inidividual local linear models, e.g. as a precompute
790         % step to running online. Note that this does not compute the
791         % Gaussian mixture regression step.
792         %
793         % gmm     - the gaussian mixture model
794         % ch      - channel structure
795         % A       - system dynamics matrix
796         % B       - input matrix
797         % C       - observation matrix
798         % D       - feedthrough matrix
799         % Q       - process noise covariance
800         % R       - measurement noise covariance
801
802         % return the state-space form of the matrices
803         d = length(ch.x);
804         K = gmm.NumComponents;
805         A = zeros(2*d,2*d,K);
806         B = zeros(2*d,1,K);
807         C = zeros(d,2*d,K);
808         D = zeros(d,1,K);
809         Q = zeros(d,d,K);
810         R = zeros(d,d,K);
811         for k = 1:K
```

```
812
813             % pull mu, sigma from the cluster
814             Mu = gmm.mu(k,:)';
815             Sigma = gmm.Sigma(:,:,k);
816
817             % compute the regression (GMR)
818             Mu_XV = Mu([ch.x ch.v]);
819             Mu_A = Mu(ch.a);
820             Mu_F = Mu(ch.f);
821             Sig_A = Sigma(ch.a,ch.a);
822             Sig_F = Sigma(ch.f,ch.f);
823             Sig_A_XV = Sigma(ch.a,[ch.x ch.v]);
824             Sig_F_XV = Sigma(ch.f,[ch.x ch.v]);
825             Sig_XV_A = Sigma([ch.x ch.v],ch.a);
826             Sig_XV_F = Sigma([ch.x ch.v],ch.f);
827             Sig_XV = Sigma([ch.x ch.v],[ch.x ch.v]);
828             Abar = Sig_A_XV * inv(Sig_XV);
829             bbar = Mu_A - Abar * Mu_XV;
830             Cbar = Sig_F_XV * inv(Sig_XV);
831             dbar = Mu_F - Cbar * Mu_XV;
832             qbar = Sig_A - Sig_A_XV * inv(Sig_XV) * Sig_XV_A;
833             rbar = Sig_F - Sig_F_XV * inv(Sig_XV) * Sig_XV_F;
834
835             % return the full state-space models
836             A(:,:,k) = [zeros(d) eye(d); Abar];
837             B(:,:,k) = [zeros(d,1); bbar];
838             C(:,:,k) = Cbar;
839             D(:,:,k) = dbar;
840             Q(:,:,k) = qbar;
```

```
841             R(:,:,k) = rbar;
842         end
843     end
844 end
845 
846 
847 % Learning
848 methods (Static)
849 
850     function noise = measurementNoise(data)
851         % noise = measurementNoise(data)
852         %
853         % This function constructs the measurement noise model from the
854         % error covariance matrices contained in data.
855         %
856         % data   - populated data structure for training
857         % noise  - signal measurement noise model
858 
859         % create the data objects
860         Ex = [];
861         Ev = [];
862         Ea = [];
863         Ef = [];
864         for k = 1:length(data)
865             Ex = [Ex; data(k).X - data(k).meas.X];
866             Ev = [Ev; data(k).V - data(k).meas.V];
867             Ea = [Ea; data(k).A - data(k).meas.A];
868             Ef = [Ef; data(k).F - data(k).meas.F];
869         end
```

```
870
871             % get the covariance
872             noise.X = cov(Ex);
873             noise.V = cov(Ev);
874             noise.A = cov(Ea);
875             noise.F = cov(Ef);
876         end
877
878         function [data, center] = centerData(data)
879             % [data, center] = centerData(data)
880             %
881             % This function computes the positional centroid of the data
882             % contained in data, and returns the centered data such that
883             % the centroid is at x = 0.
884             %
885             % data   - populated data structure for training
886             % center - positional center of the data
887             % data   - centered data structure
888
889             % create the data matrix
890             D = [];
891             for k = 1:length(data)
892                 [Dn, ch] = knet.processData(data(k));
893                 D = [D; Dn(data(k).id_cluster, :)];
894             end
895             D = D(:,ch.x);
896
897             % compute mean and covariance of position
898             D_mu = mean(D);
```

```
899             D_cov = cov(D);
900             center.Mu = D_mu;
901             center.Sigma = D_cov;
902 
903             % now center the data
904             for k = 1:length(data)
905                 data(k).X = data(k).X - D_mu;
906                 data(k).meas.X = data(k).meas.X - D_mu;
907             end
908         end
909 
910         function gmm = clusterData(data, K, covMode, reg)
911             % gmm = clusterData(data, K, mode, reg)
912             %
913             % This function clusters the data indices defined by
914             % data.id_cluster into target states.
915             %
916             % data   - populated data structure for training
917             % K      - number of clusters
918             % mode   - 'diagonal' or 'full'
919             % reg    - regularization on the covariance
920             % gmm    - learned target gmm
921             fprintf('Clustering data with K = %i...\n',K)
922 
923             % create the data matrix
924             Dtrain = [];
925             for k = 1:length(data)
926                 [Dn, ch] = knet.processData(data(k));
927                 Dtrain = [Dtrain; Dn(data(k).id_cluster, :)];
```

```
928             end
929             vars = [ch.x ch.v ch.a ch.f];
930             Dtrain = Dtrain(:,vars);
931
932             % GMM learning options
933             % www.mathworks.com/help/stats/tune-gaussian-mixture-models.html
934             % www.mathworks.com/help/stats/fitgmdist.html
935             options = statset('MaxIter',10000,'Display','off');
936             gmm = fitgmdist(Dtrain,K,...
937                 'CovarianceType',covMode,...
938                 'Replicates',10,...
939                 'SharedCovariance',false,...
940                 'RegularizationValue',reg,...
941                 'Options',options);
942         end
943
944         function btree = linkClusters(gmm, method)
945             % btree = linkClusters(gmm, method)
946             %
947             % This function constructs an agglomerative hierarchy of the
948             % cluster model in gmm by performing a linkage on the GMM
949             % centers.
950             %
951             % gmm    - learned target gmm
952             % method - linking distance metric, options are: 'euclidean',
953             %          'bhattacharyya', and 'hellinger'
954             % btree  - the output binary tree (see linkage for more info)
955             fprintf('Building GMM hierarchy...\n')
956             if nargin < 2
```

```
957             method = 'euclidean';
958         end
959
960         % compute the distance measure
961         vars = [1:8];
962         n = gmm.NumComponents;
963         D = zeros(1,(n^2 - n)/2);
964         k = 1;
965         for i = 1:n
966             for j = i+1:n
967                 if strcmpi(method, 'euclidean')
968                     D(k) = sqrt(sum((gmm.mu(j,vars) - gmm.mu(i,vars)).^2));
969                 elseif strcmpi(method, 'bhattacharyya')
970                     Sigma = (gmm.Sigma(vars,vars,j) +...
971                         gmm.Sigma(vars,vars,i))/2;
972                     e = gmm.mu(j,vars) - gmm.mu(i,vars);
973                     sig0 = log(det(Sigma) /...
974                         (sqrt(det(gmm.Sigma(vars,vars,j)) *...
975                         det(gmm.Sigma(vars,vars,i))))) / 2;
976                     D(k) = e * inv(Sigma) * e'/8 + sig0;
977                 elseif strcmpi(method, 'hellinger')
978                     Sigma = (gmm.Sigma(vars,vars,j) +...
979                         gmm.Sigma(vars,vars,i))/2;
980                     e = gmm.mu(j,vars) - gmm.mu(i,vars);
981                     sig0 = (det(gmm.Sigma(vars,vars,j))^(.25) *...
982                         det(gmm.Sigma(vars,vars,i))^(.25)) /...
983                         (det(Sigma)^(.5));
984                     H2 = 1 - sig0 * exp(-e * inv(Sigma) * e'/8);
985                     D(k) = sqrt(H2);
```

```
986                 end
987                 k = k + 1;
988             end
989         end
990
991         % now compute the linkage
992         btree = linkage(D);
993     end
994
995     function [states, rank] = findStates(data, gmm, verbose)
996         % [states, rank] = findStates(data, gmm, verbose)
997         %
998         % This function ranks candidate states in the data
999         %
1000        % data    - populated data structure for training
1001        % gmm     - learned target gmm
1002        % verbose - flag for verbose print
1003        % states - sorted state indices
1004        % rank    - state rank (dot produce Eqn 5.47)
1005
1006        if nargin < 3
1007            verbose = 0;
1008        end
1009
1010        % push the data and cluster
1011        D = [];
1012        I = [];
1013        for k = 1:length(data)
1014            % get the data
```

```
1015            [Dn, ch] = knet.processData(data(k));
1016            vars = [ch.x ch.v ch.a ch.f];
1017            id = data(k).id_cluster;
1018            D = [D; Dn(id,vars)];
1019            I = [I; gmm.cluster(Dn(id,vars))];
1020        end
1021
1022        % compute the vector field divergence within each cluster
1023        for i = 1:gmm.NumComponents
1024            id = find(I == i);
1025
1026            % compute the dot product of the data with the cluster
1027            % this gives a measure of motion divergence
1028            vars = [ch.v ch.a];
1029            dp(i) = 0;
1030            ct(i) = 0;
1031            ns = length(id);
1032            for j = 1:length(id)
1033                dp(i) = dp(i) + dot(D(id(j),vars),gmm.mu(i,vars));
1034            end
1035            dp(i) = dp(i) / ns;
1036        end
1037        [rank, states] = sort(dp, 'ascend');
1038
1039        % print out
1040        if (verbose)
1041            fprintf('S = {');
1042            for k = 1:length(states)
1043                fprintf(' %i ',states(k));
```

```
1044            end
1045            fprintf('}\n');
1046        end
1047    end
1048
1049    function cSeq = clusterSequence(data, gmm, optn)
1050        % cSeq = findclusterSequence(data, gmm)
1051        %
1052        % This function finds the sequence of gmm's from data
1053        %
1054        % data     - the populated data structure
1055        % gmm      - the gmm model
1056        % figNumber - (optional) figure number for plot
1057        % cSeq     - structure of cluster sequence
1058        fprintf('Finding the sequence of clusters through data...\n')
1059
1060        if nargin == 3
1061            figure(figNumber),clf
1062            cols = ceil(sqrt(length(data)));
1063            rows = ceil(length(data)/cols);
1064            Nslots = rows*cols;
1065        end
1066
1067        % for each element of data, compute the sequence
1068        Time = [];
1069        Index = [];
1070        Trial = [];
1071        Cluster = []; % state activation
1072        Likelihood = []; % state likelihood
```

```
1073            Posterior = []; % state posterior
1074            S0 = zeros(gmm.NumComponents,1); % start state probability
1075            for k = 1:length(data)
1076
1077                % get the data
1078                [Dn, ch] = knet.processData(data(k));
1079                vars = [ch.x ch.v ch.a ch.f];
1080                id = data(k).id_cluster;
1081                id = sort(id);
1082                Dn = Dn(id,:);
1083
1084                % find the cluster indices (while this works well, it
1085                % has a little bit of noise... maybe we can be a little
1086                % smarter about this, i.e. add hysteresis into the switch
1087                Ci = gmm.cluster(Dn(:,vars));
1088                Po = gmm.posterior(Dn(:,vars));
1089                Li = [];
1090                for i = 1:gmm.NumComponents
1091                    Li(:,i) = mvnpdf(Dn(:,vars),...
1092                        gmm.mu(i,:), gmm.Sigma(:,:,i));
1093                end
1094
1095                % push state posterior
1096                Cl = Po == max(Po,[],2)*ones(1,size(Po,2));
1097
1098                % append the full information
1099                Trial = [Trial; k*ones(length(Cl),1)];
1100                Cluster = [Cluster; Cl];
1101                Likelihood = [Likelihood; Li];
```

```
1102            Posterior = [Posterior; Po];
1103            S0 = S0 + Po(1,:)'./length(data);
1104
1105            if nargin == 3
1106                subplot(rows,cols,k)
1107                imagesc(Po')
1108                xlabel('Sample')
1109                ylabel('Cluster')
1110            end
1111
1112            % find the state start and end
1113            state_s = find([1; abs(diff(Ci))] > 0);
1114            state_f = find([abs(diff(Ci)); 1] > 0);
1115
1116            % compute the one-hot activation
1117            T0 = Dn(state_s, ch.t);
1118            Tf = Dn(state_f, ch.t);
1119
1120            Time = [Time; Tf-T0];
1121            Index = [Index; Ci(state_s)];
1122        end
1123
1124        % Compute the stochastic transition matrix
1125        % http://pi.math.cornell.edu/~mec/Winter2009/\
1126        % RalucaRemus/Lecture2/lecture2.html
1127        Y = Cluster(2:end,:)';
1128        X = Cluster(1:end-1,:)';
1129        A = (Y*X') / (X*X'); % min Frobenius norm
1130        B = (X*Y') / (Y*Y'); % min Frobenius norm
```

```
1131
1132            % Compute the stochastic adjacency matrices
1133            Wa = (A - diag(diag(A))) ./ sum(A - diag(diag(A)));
1134            Wb = (B - diag(diag(B))) ./ sum(B - diag(diag(B)));
1135
1136            % lastly, find the connectivity in the state space
1137            Connectivity = ones(size(A));
1138            for i = 1:gmm.NumComponents
1139                % get i^th distance
1140                xv_i = gmm.mu(i,[ch.x ch.v]);
1141                v_i = gmm.mu(i,[ch.v]);
1142                a_i = gmm.mu(i,[ch.a]); % acceleration projection
1143
1144                % compute the vector field projection onto the other points
1145                va_i = [v_i, a_i]; % vector field projection (v, a)
1146                for j = 1:gmm.NumComponents
1147                    xv_j = gmm.mu(j,[ch.x ch.v]);
1148                    va_j(j,:) = xv_j - xv_i; % error between points (x, v)
1149
1150                    % compute the projection weight
1151                    den = (norm(va_i,2) * norm(va_j(j,:),2));
1152                    Connectivity(i,j) = dot(va_i, va_j(j,:)) ./ den;
1153                    if den == 0
1154                        Connectivity(i,j) = 1;
1155                    end
1156                    Connectivity(i,j) = (pi - acos(Connectivity(i,j))) ./ pi;
1157                end
1158            end
1159
```

```
1160            % compute mean/max times spent in states
1161            for i = 1:gmm.NumComponents
1162                id = find(Index == i);
1163                Tmax(i) = max(Time(id));
1164                Tmean(i) = mean(Time(id));
1165                Tvar(i) = var(Time(id));
1166            end
1167
1168            % push sequence information to structure
1169            cSeq.Tmax = Tmax;
1170            cSeq.Tmean = Tmean;
1171            cSeq.Tvar = Tvar;
1172            cSeq.Connectivity = Connectivity; % 'Connectivity'
1173            cSeq.FwdStochastic = A;
1174            cSeq.BwdStochastic = B;
1175            cSeq.FwdAdjacency = Wa;
1176            cSeq.BwdAdjacency = Wb;
1177            cSeq.Start = S0;
1178            cSeq.Trial = Trial;
1179            cSeq.FullPosterior = Posterior;
1180            cSeq.FullCluster = Cluster;
1181            cSeq.FullLikelihood = Likelihood;
1182        end
1183
1184        function [actions, Pa] = findActions(Wa, Wb, states, verbose)
1185            % actions = findActions(Wa, Wb, states, verbose)
1186            %
1187            % Given a stochastic transition matrix A, find all the unique
1188            % paths between state indices in 'States', i.e.:
```

```
1189            %
1190            % The set of unique paths {s_1,...s_k}_i^n
1191            %
1192            % The matrix A defines:
1193            %       P(s_k+1) = A P(s_k)
1194            %
1195            % This function returns the unique paths between States
1196            % through the matrix A, using breadth-first search. States must
1197            % be specified as indices. The first state must be the
1198            % starting state, and the last state must be the ending state
1199            %
1200            % A      - the stochastic transition matrix
1201            % States - the vector of indices
1202
1203            % set verbose flag
1204            if nargin < 4
1205                verbose = 0;
1206            end
1207
1208            % open sets
1209            openPaths = num2cell(states); % FIFO queue of paths
1210            closedPaths = {};             % set of closed paths
1211
1212            % breadth-first search
1213            while ~isempty(openPaths)
1214
1215                % denqueue
1216                path = openPaths{end};
1217                openPaths = openPaths(1:end-1);
```

```
1218
1219            % roll out next state
1220            i = path(end);
1221            S = zeros(length(Wa),1);
1222            S(i) = 1;
1223            Snext = Wa * S;
1224            j = find(Snext > 0); % Pr(j|i)
1225
1226            % for each node...
1227            for k = 1:numel(j)
1228
1229                % if we haven't already visited this state...
1230                if (sum(path == j(k)) == 0)
1231
1232                    % if the node is a state, close it...
1233                    if sum(j(k) == states) > 0
1234
1235                        % close the path
1236                        closedPaths = [closedPaths {[path j(k)]}];
1237
1238                    % if the node is open...
1239                    else
1240
1241                        % keep the path open
1242                        openPaths = [openPaths {[path j(k)]}];
1243                    end
1244                end
1245            end
1246        end
```

```
1247            actions = closedPaths;
1248
1249            % return the probablistic rank of each action
1250            Pa = zeros(1,length(closedPaths));
1251            for k = 1:length(closedPaths)
1252                S0 = zeros(length(Wa),1);
1253                Sf = zeros(length(Wa),1);
1254                S0(closedPaths{k}(1)) = 1;
1255                Sf(closedPaths{k}(end)) = 1;
1256                ns = length(closedPaths{k})-1;
1257                S0f = Wa^(ns) * S0; % forward
1258                Sf0 = Wb^(ns) * Sf; % backward
1259                Pa(k) = S0f(closedPaths{k}(end)) * Sf0(closedPaths{k}(1));
1260            end
1261
1262            % Now, find the action manifolds (i.e. all the clusters
1263            % occuring in between similar states)
1264            alist = [];
1265            for k = 1:length(closedPaths)
1266                alist = [alist; closedPaths{k}([1 end])];
1267                clist{k} = closedPaths{k}(2:end-1);
1268            end
1269            [C,IA,IC] = unique(alist,'rows');
1270
1271            % for each unique action, combine
1272            P = [];
1273            for k = 1:length(C)
1274
1275                % grab action subset
```

```
1276            id = find(IC == k);
1277
1278            % accumulate the clusters in the middle
1279            mc =[];
1280            for i = 1:length(id)
1281                mc = [mc, clist{id(i)}];
1282            end
1283            mc = unique(mc); % find unique middles
1284
1285            % unique actions & their probabilities
1286            P(k) = sum(Pa(id));
1287            actions{k} = [C(k,1) mc C(k,2)];
1288        end
1289        P = P ./ sum(P);
1290
1291        % sort (descending)
1292        [¬,i] = sort(P, 'descend');
1293        actions = actions(i);
1294        P = P(i);
1295
1296        % print out
1297        if (verbose)
1298            for k = 1:length(actions)
1299                fprintf('Pa = %.2f | A = {',P(k));
1300                for j = 1:length(actions{k})
1301                    fprintf(' %i ',actions{k}(j));
1302                end
1303                fprintf('}\n');
1304            end
```

```
1305            fprintf('\n');
1306        end
1307    end
1308
1309    function aSeq = actionSequence(data, gmm, A, S, showPlot)
1310        % aSeq = actionSequence(data, gmm, A, S, showPlot)
1311        %
1312        % This function finds the action sequence in data
1313        %
1314        % data     - the populated data structure
1315        % gmm      - the gmm model
1316        % A        - actions
1317        % S        - states
1318        % showPlot - (optional) flag to show plot
1319        % aSeq     - structure of action sequence
1320
1321        if nargin < 5
1322            showPlot = false;
1323        end
1324
1325        % push the data into Psi matrix
1326        Psi = [];
1327        trial = [];
1328        work = [];
1329        time = [];
1330        for k = 1:length(data)
1331            [D, ch] = knet.processData(data(k));
1332            id = data(k).id_cluster;
1333            id = sort(id);
```

```
1334            vars = [ch.x ch.v ch.a ch.f];
1335            Psi = [Psi; D(id,vars)];
1336            trial = [trial; k*ones(length(id),1)];
1337
1338            % compute the work and time
1339            Delta = [0; D(id(2:end),ch.t)-D(id(1:end-1),ch.t)];
1340            Energy = dot(-D(id,ch.f),D(id,ch.v),2).*Delta;
1341            work = [work; cumsum(Energy)];
1342        end
1343        in = [ch.x ch.v];
1344        out = [ch.a ch.f];
1345
1346        % compute the cluster posterior (x, v, f) data
1347        clusterPost = knet.cluster(gmm, Psi(:,[in out]), [in out]);
1348
1349        % produce the state posterior and activation from data
1350        statePost = clusterPost(:,S);
1351        stateActivation = statePost > 0.5;
1352
1353        % find the cluster index
1354        stateIndex = zeros(length(statePost),1);
1355        for i = 1:length(S)
1356            id = find(stateActivation(:,i) == 1);
1357            stateIndex(id) = i;
1358        end
1359
1360        % get the mahalanobis distance to the state
1361        stateMahal = zeros(size(Psi,1),length(S));
1362        for i = 1:length(S)
```

```
1363            s = S(i);
1364            [¬,stateMahal(:,i)] = knet.gaussian(Psi(:,in)',...
1365                gmm.mu(s,in)',gmm.Sigma(in,in,s));
1366        end
1367
1368        % remove zero regions and keep track of indices
1369        stateIdx = find(sum(stateActivation,2)>0);
1370        stateAct = stateActivation(stateIdx,:);
1371
1372        % find the state start and end
1373        state_s = stateIdx(find([1; sum(diff(stateAct)>0,2)]));
1374        state_f = stateIdx(find([sum(diff(stateAct)<0,2); 1]));
1375        state_m = zeros(size(state_s));
1376
1377        % show the activation states and start/stop indices
1378        for i = 1:length(state_s)
1379            si = stateIndex(state_s(i)); % state index
1380            id = state_s(i):state_f(i); % indices of region
1381            state_m(i) = find(min(stateMahal(id,si)) ==...
1382                stateMahal(id,si),1,'first')+state_s(i)-1;
1383        end
1384
1385        % compute the state visitation
1386        stateVisit = zeros(size(stateActivation));
1387        for i = 1:length(S)
1388            stateVisit(state_s(stateIndex(state_s)==i),i) = 1;
1389        end
1390        for i = 1:max(trial)
1391            id = trial == i;
```

```
1392            stateVisit(id,:) = cumsum(stateVisit(id,:));
1393        end
1394
1395        % get the state indices
1396        si = [stateIndex(state_s(1:end-1)),...
1397              stateIndex(state_s(2:end))];
1398
1399        % get the action indices
1400        actionIndex = zeros(size(si,1),1);
1401        ai = zeros(size(si,1)-1,2);
1402        for a = 1:length(A)
1403            % what happens if this condition isn't true?
1404            id = (A{a}(1) == S(si(:,1))) & (A{a}(end) == S(si(:,2)));
1405            actionIndex(id) = a;
1406            ai(id,:) = [state_m(find(id)), state_m(find(id)+1)-1];
1407        end
1408
1409        % remove zeros
1410        ai(actionIndex == 0,:) = [];
1411        actionIndex(actionIndex == 0) = [];
1412
1413        % generate the action activation
1414        actionClass = zeros(size(trial,1),1);
1415        actionActivation = zeros(size(trial,1),length(A));
1416        for i = 1:length(actionIndex)
1417            actionActivation(ai(i,1):ai(i,2),actionIndex(i)) = 1;
1418            actionClass(ai(i,1):ai(i,2),1) = actionIndex(i);
1419            xm = stateMahal(state_m(i+1),:);
1420        end
```

```
1421
1422            % compute the action visitation
1423            actionVisit = zeros(size(actionActivation));
1424            for i = 1:length(A)
1425                actionVisit(ai(actionIndex==i,1),i) = 1;
1426            end
1427            for i = 1:max(trial)
1428                id = trial == i;
1429                actionVisit(id,:) = cumsum(actionVisit(id,:));
1430            end
1431
1432            if showPlot
1433                figure(3),clf
1434                colormap(gcf,flipud(hot))
1435                h(1)=subplot(211);
1436                hold on; box on;
1437                imagesc(log(stateActivation' .* stateMahal'))
1438                plot(state_m, stateIndex(state_m),'+b') % mahal point
1439                line([state_m'; state_m'],...
1440                    (ones(length(state_m),1) * [0 length(S)+1])',...
1441                    'LineStyle',':','Color','b')
1442                ylabel('State')
1443                set(gca,'XTick',[])
1444                set(gca,'YTick',[])
1445                axis tight
1446                h(2)=subplot(212);
1447                hold on; box on;
1448                imagesc(actionActivation')
1449                line([state_m'; state_m'],...
```

```
1450                (ones(length(state_m),1) * [0 length(A)+1])',...
1451                'LineStyle',':','Color','b')
1452            ylabel('Action')
1453            set(gca,'XTick',[])
1454            set(gca,'YTick',[])
1455            axis tight
1456            xlabel('Sample i = $\{1,\ldots,n_s\}$')
1457            linkaxes(h,'x')
1458        end
1459
1460        % push to action sequence struct
1461        aSeq.StateIndices = [state_s, state_m, state_f];
1462        aSeq.StateClass = stateIndex(state_s);
1463        aSeq.ActionIndices = ai;
1464
1465        aSeq.TrialFull = trial;
1466        aSeq.StateMahalFull = stateMahal;
1467        aSeq.StateVisitFull = stateVisit;
1468        aSeq.ActionActivationFull = actionActivation;
1469        aSeq.ActionVisitFull = actionVisit;
1470        aSeq.ActionClassFull = actionClass;
1471        aSeq.WorkFull = work;
1472
1473        % get index of relevant states
1474        idS = ai(:,1);
1475        idF = ai(:,2);
1476        aSeq.Trial = trial(idS);
1477        aSeq.StateMahal = stateMahal(idF,:);
1478        aSeq.StateVisit = stateVisit(idF,:);
```

```
1479            aSeq.ActionActivation = actionActivation(idF,:);
1480            aSeq.ActionVisit = actionVisit(idF,:);
1481            aSeq.ActionClass = actionClass(idF,:);
1482            aSeq.Work = work(idF,:);
1483        end
1484
1485        function transition = learnSequence(nlayers, aSeq, A, S, genFcn,...
1486            noptn, verbose, figNumber)
1487            % transition = learnSequence(nlayers, aSeq, A, S, genFcn,...
1488            %   noptn, verbose, figNumber)
1489            %
1490            % This function learns an autoregressive neural network that
1491            % predicts the action sequence
1492            %
1493            % nlayers   - the neural network hidden layer sizes
1494            % aSeq      - action sequence data structure
1495            % A         - actions
1496            % S         - states
1497            % genFcn    - flag to generate function
1498            % verbose   - flag for verbose outputs
1499            % noptn     - network options
1500            % figNumber - (optional)
1501            % transition - the learned transition model
1502            %
1503            % If noptn is specified, it must be a structure that contains
1504            %   .trainRatio
1505            %   .valRatio
1506            %   .testRatio
1507            %   .transferFcn
```

```
1508            %   .regularization
1509
1510            % learn the transition model - use the mahalanobis distance to
1511            % the target state, while that state is active to produce a
1512            % lognormal distribution for the acitvation regions.
1513            for a = 1:length(A)
1514                s = find(A{a}(end) == S);
1515                id = aSeq.StateIndices(s == aSeq.StateClass,:);
1516                d = [];
1517                for j = 1:length(id)
1518                    % grab beginning to middle to (3) would be end
1519                    idx = id(j,1):id(j,3);
1520                    d = [d; aSeq.StateMahalFull(idx,s)];
1521                end
1522                % prior to this, I used the closest mahalanobis distance
1523                % while in the target state to compute the probability of
1524                % transition, which led to a brittle system. Now, we use
1525                % all the data while the state is active to generate our
1526                % lognormal transition distribution
1527                ptrans(a) = fitdist(d,'lognormal');
1528            end
1529            transition.ptrans = ptrans;
1530
1531            % get the start/end actions
1532            A0 = zeros(1,length(A));
1533            Af = zeros(1,length(A));
1534            for k = 1:max(aSeq.Trial)
1535                id = find(aSeq.Trial == k);
1536                if ¬isempty(id)
```

```
1537                A0 = A0 + aSeq.ActionActivation(id(1),:);
1538                Af = Af + aSeq.ActionActivation(id(end),:);
1539            end
1540        end
1541        A0 = A0 ./ sum(A0);
1542        Af = Af ./ sum(Af);
1543        transition.FirstAction = A0;
1544        transition.LastAction = Af;
1545
1546        % get the activation matrices
1547        Y = aSeq.ActionActivationFull(2:end,:)';
1548        X = aSeq.ActionActivationFull(1:end-1,:)';
1549
1550        % get the stochastic forward transition matrix
1551        % This is the min frobenius norm of e.g.: Y = A X
1552        Wa = (Y*X') / (X*X');
1553        Wa = Wa ./ sum(Wa,1);
1554        Wb = (X*Y') / (Y*Y');
1555        Wb = Wb ./ sum(Wb,1);
1556        transition.FwdStochastic = Wa;
1557        transition.BwdStochastic = Wb;
1558
1559        % make sure cols/rows sum to 1
1560
1561        % Partition the training data (use work to avoid activation
1562        % problem - nonlinear switching)
1563        XTrain = [aSeq.Work(1:end-1,:),...
1564                  aSeq.ActionActivation(1:end-1,:)]';
1565        YTrain = [aSeq.ActionActivation(2:end,:)]';
```

```
1566
1567            % Create a Pattern Recognition Network
1568            net = patternnet(nlayers);
1569            if ¬isempty(noptn)
1570                net.performParam.regularization = noptn.regularization;
1571                net.divideParam.trainRatio = noptn.trainRatio;
1572                net.divideParam.valRatio = noptn.valRatio;
1573                net.divideParam.testRatio = noptn.testRatio;
1574                %'purelin', 'logsig', 'tansig'
1575                net.layers{1}.transferFcn = noptn.transferFcn;
1576            else
1577                net.divideParam.trainRatio = 0.7;
1578                net.divideParam.valRatio = 0.3;
1579                net.divideParam.testRatio = 0;
1580            end
1581
1582            % choose output options
1583            if (verbose > 2)
1584                net.trainParam.showWindow = true;
1585                net.trainParam.showCommandLine = false;
1586            elseif (verbose > 1)
1587                net.trainParam.showWindow = false;
1588                net.trainParam.showCommandLine = true;
1589            else
1590                net.trainParam.showWindow = false;
1591                net.trainParam.showCommandLine = false;
1592            end
1593
1594            % train
```

```
1595            [net,tr] = train(net,XTrain,YTrain);
1596            net.name = 'kNet action activation network';
1597
1598            % Prediction
1599            YPred = net(XTrain);
1600
1601            % return the learned transition model
1602            transition.net = net;
1603            transition.fcnName = [];
1604            if genFcn
1605                transition.fcnName = 'transitionNet';
1606                genFunction(net, transition.fcnName,...
1607                    'MatrixOnly', 'yes', 'ShowLinks', 'yes');
1608            end
1609
1610            % plot the learned sequence results
1611            if ((nargin > 7) && (figNumber > 0))
1612
1613                % plot the neural network
1614                figure(figNumber),clf
1615                h(1)=subplot(311);
1616                imagesc(XTrain);
1617                colormap(h(1),hot)
1618                ylabel('Inputs $\{a_i,W_i\}$')
1619                xlabel('Sample $i=\{1,\ldots,n_s-1\}$')
1620                set(gca,'XTick',[])
1621                set(gca,'YTick',[])
1622                h(2)=subplot(312);
1623                imagesc(YTrain);
```

```
1624            colormap(h(2),flipud(bone))
1625            ylabel('Truth $a_{i+1}$')
1626            xlabel('Sample $i=\{2,\ldots,n_s\}$')
1627            set(gca,'XTick',[])
1628            set(gca,'YTick',[])
1629            h(3)=subplot(313);
1630            imagesc(YPred);
1631            colormap(h(3),flipud(bone))
1632            ylabel('Prediction $p(a_{i+1})$')
1633            xlabel('Sample $i=\{2,\ldots,n_s\}$')
1634            set(gca,'XTick',[])
1635            set(gca,'YTick',[])
1636            linkaxes(h,'x')
1637
1638            % plot the transition model
1639            figure(figNumber+1),clf
1640            Na = length(A);
1641            cols = ceil(sqrt(Na));
1642            rows = ceil(Na/cols);
1643
1644            xt = logspace(-4,4,1001);
1645            for a = 1:Na
1646                subplot(rows,cols,a)
1647                ylabel(sprintf('$a_%i$',a))
1648                hold on;
1649                s = find(A{a}(end) == S);
1650                id = aSeq.StateIndices(s == aSeq.StateClass,:);
1651                d = [];
1652                for j = 1:length(id)
```

```
1653                    idx = id(j,1):id(j,3);
1654                    d = [d; aSeq.StateMahalFull(idx,s)];
1655                end
1656                histogram(log(d),151,...
1657                    'Normalization','pdf',...
1658                    'FaceColor',[.2 .2 .2],...
1659                    'EdgeColor','none')
1660                pt = ptrans(a).cdf(xt);
1661                plot(log(xt),1-pt,'-b','LineWidth',1.5)
1662                xlim([-4 4])
1663                ylim([0 1.1])
1664                set(gca,'YTick',[0 1])
1665                set(gca,'XTick',[])
1666                legend('Data','Fit CDF')
1667                xlabel('$\ln(D_M(s^*_a))$')
1668            end
1669        end
1670
1671        % get peformance (cross-entropy)
1672        transition.net_training = tr;
1673    end
1674
1675    function traj = learnActionTrajectories(gmm, A, ch, dt, h)
1676        % traj = learnActionTrajectories(gmm, A, ch, dt, h)
1677        %
1678        % This function learns trajectories for each action in A
1679        %
1680        % gmm       - the gmm model
1681        % A         - actions
```

```
1682            % ch        - channel structure
1683            % dt        - sample rate for DS monte carlo sampling
1684            % h         - handle to the figure for display
1685            % traj      - structure of optimized trajectories
1686
1687            % show guess on plot
1688            plotFigs = false;
1689            if nargin > 4
1690                plotFigs = true;
1691                figure(h);
1692                hold on;
1693                sp = scatter(0,0,5,'filled');
1694                pp = plot(0,0,'-r','LineWidth',4);
1695            end
1696
1697            % Learn each trajectory
1698            for a = 1:length(A)
1699
1700                % get the sequence
1701                manifold = A{a};
1702
1703                % channels
1704                xi = [ch.x ch.v];
1705
1706                % start/stop states
1707                xv1 = gmm.mu(manifold(1),xi);
1708                xvf = gmm.mu(manifold(end),xi);
1709
1710                % We need to start with a decent guess, so we'll sample
```

```
1711            % from the start state, and inntegrate until we stop
1712            % getting closer to the target state. The closest particle
1713            % trajectory initializes the algorithm
1714            np = 5000;
1715            Xp = knet.sample(gmm, manifold(1), np);
1716            Xp = Xp(:,xi);
1717            Mp = Inf(np,1); % min mahal dist
1718            if plotFigs
1719                sp.XData = Xp(:,ch.x(1));
1720                sp.YData = Xp(:,ch.v(1));
1721            end
1722
1723            % Start the loop
1724            i = 2;
1725            done = false;
1726            m_vec = Inf;
1727            while ¬done
1728
1729                % integrate forward
1730                Xp(:,:,i) = knet.rk4singlestep(...
1731                    @(t,x)knet.predictMotion(t, x, manifold, gmm, ch),...
1732                    dt, 0, Xp(:,:,i-1));
1733
1734                % get mahalanobis distance (to target)
1735                [¬, M] = knet.gaussian(Xp(:,:,i)',...
1736                    gmm.mu(manifold(end), xi)',...
1737                    gmm.Sigma(xi,xi,manifold(end)));
1738
1739                % update min mahal dist
```

```
1740                Mp = min(Mp, M');
1741                m_vec(i) = sum(Mp);
1742
1743                % stop if time > 1
1744                % if we haven't gotten smaller, that's it
1745                done = m_vec(i) == m_vec(i-1);
1746
1747                % show
1748                if plotFigs
1749                    sp.XData = Xp(:,ch.x(1),i);
1750                    sp.YData = Xp(:,ch.v(1),i);
1751                    sp.CData = Mp;
1752                end
1753
1754                % update index
1755                i = i + 1;
1756            end
1757
1758            % find the constraint violation of the particles
1759            Cp = zeros(np,1);
1760            for j = 1:np
1761
1762                % get the trajectory
1763                xv = squeeze(Xp(j,:,:));
1764
1765                % truncate the ends
1766                [¬, M0] = knet.gaussian(xv, gmm.mu(manifold(1), xi)',...
1767                        gmm.Sigma(xi,xi,manifold(1)));
1768                n0 = find(M0 == min(M0));
```

```
1769                [¬, Mf] = knet.gaussian(xv, gmm.mu(manifold(end), xi)',...
1770                        gmm.Sigma(xi,xi,manifold(end)));
1771                nf = find(Mf == min(Mf));
1772                nt = nf - n0 + 1;
1773                xv = xv(:,n0:nf);
1774
1775                % compute cost
1776                theta = [xv(:); dt];
1777                if ¬isempty(xv)
1778                    [¬,c] = knet.constraintTrajectory(theta,...
1779                        gmm, manifold([1 end]), ch);
1780                else
1781                    c = Inf;
1782                end
1783                Cp(j) = c;
1784            end
1785
1786            % find the 'winning' trajectory
1787            iwin = find(Cp == min(Cp),1,'first');
1788
1789            % select the trajectory
1790            xv = squeeze(Xp(iwin,:,:));
1791
1792            % find the ends
1793            [¬, M0] = knet.gaussian(xv, gmm.mu(manifold(1), xi)',...
1794                gmm.Sigma(xi,xi,manifold(1)));
1795            n0 = find(M0 == min(M0));
1796            [¬, Mf] = knet.gaussian(xv, gmm.mu(manifold(end), xi)',...
1797                gmm.Sigma(xi,xi,manifold(end)));
```

```
1798            nf = find(Mf == min(Mf));
1799            nt = nf - n0 + 1;
1800            xv = xv(:,n0:nf);
1801
1802            % show on plot
1803            if plotFigs
1804                pp.XData = xv(ch.x(1),:);
1805                pp.YData = xv(ch.v(1),:);
1806                drawnow
1807            end
1808
1809
1810            % Now we begin the optimization routine, using nonlinear
1811            % function minimization with constraints.
1812
1813            % initial parameters
1814            theta0 = [xv(:); dt];
1815
1816            % optimize
1817            options = optimoptions('fmincon','Display','iter',...
1818                'PlotFcn','optimplotfval',...
1819                'MaxFunctionEvaluations',nt*length(xi)*1e3,...
1820                'ConstraintTolerance',1e-6);
1821
1822            thetaStar = fmincon(...
1823                @(x)knet.costTrajectory(x, manifold, gmm, ch),theta0,...
1824                    [],[],[],[],[],[],...
1825                @(x)knet.constraintTrajectory(x, gmm,...
1826                    manifold([1 end]), ch),...
```

```
1827                options);
1828
1829            % get the trajectory
1830            xvStar = thetaStar(1:end-1);
1831            dtStar = thetaStar(end);
1832            xvStar = reshape(xvStar, [length(xi) nt]);
1833
1834            traj(a).t = (0:nt-1)*dtStar;  % time
1835            traj(a).x = xvStar(1:length(ch.x),:);  % position
1836            traj(a).v = xvStar(length(ch.x)+1:end,:); % velocity
1837
1838            % update plot
1839            if plotFigs
1840                pp.XData = traj(a).x(1,:);
1841                pp.YData = traj(a).v(1,:);
1842                pause(3)
1843            end
1844        end
1845    end
1846
1847    function [Ra, Rf] = modelUncertainty(data, gmm, A, aseq)
1848        % [Ra, Rf] = modelUncertainty(data, gmm, A, aseq)
1849        %
1850        % This function returns the model covariance on the training
1851        % data. It computes the prediction from UNFILTERED measurement
1852        % data to get the model uncertainty for runtime.
1853        %
1854        % data     - the populated data structure
1855        % gmm      - the gmm model
```

```
1856            % A        - actions
1857            % aSeq     - action sequence
1858            % Ra       - acceleration covariance
1859            % Rf       - force covariance
1860
1861            % action class
1862            ac = aseq.ActionClass;
1863            ai = aseq.ActionIndices;
1864
1865            % get the XV input data
1866            XV = [];
1867            A_meas = [];
1868            F_meas = [];
1869            for k = 1:length(data)
1870                [¬, ch] = knet.processData(data(k));
1871                id = data(k).id_cluster;
1872                id = sort(id);
1873                xv = [data(k).meas.X(id,:), data(k).meas.V(id,:)];
1874                XV = [XV; xv];
1875                A_meas = [A_meas; data(k).meas.A(id,:)];
1876                F_meas = [F_meas; data(k).meas.F(id,:)];
1877            end
1878
1879            % predict the mean values
1880            A_mu = zeros(size(A_meas));
1881            F_mu = zeros(size(F_meas));
1882            for a = 1:length(A)
1883                man = A{a};
1884                aindex = ai(ac == a,:);
```

```
1885                for i = 1:length(aindex)
1886                    id = aindex(i,1):aindex(i,2);
1887                    A_mu(id,:) = knet.predictAccel(0, XV(id,:), man, gmm, ch);
1888                    F_mu(id,:) = knet.predictForce(0, XV(id,:), man, gmm, ch);
1889                end
1890            end
1891
1892            % get the covariance
1893            Ra = cov(A_mu - A_meas);
1894            Rf = cov(F_mu - F_meas);
1895        end
1896
1897        function model = train(data, ti, vi, K, N, M, clusterMode,...
1898            includeSeq, verbose)
1899            % model = train(data, ti, vi, K, N, M, clusterMode,...
1900            %   includeSeq, verbose)
1901            %
1902            % This function trains the GMM and neural network given a
1903            % number of parameter choices.
1904            %
1905            % data       - the populated data structure
1906            % ti         - indices in data to be used for training
1907            % vi         - indices in data to be used for validation
1908            % K          - number of clusters
1909            % N          - number of states (can leave empty if unknown)
1910            % M          - number of neurons
1911            % clusterMode - only supports 'full'
1912            % includeSeq - flag to eval sequence prediction in likelihood
1913            % verbose    - set high to display outputs
```

```
1914            % model      - a cell of models and performance
1915
1916            % initial an empty model list
1917            model = {};
1918
1919            % set the GMM
1920            gmm = knet.clusterData(data(ti), K, clusterMode, 1e-5);
1921
1922            % find sequence of gmm clusters
1923            cseq = knet.clusterSequence(data(ti), gmm);
1924            Wa = cseq.FwdAdjacency;
1925            Wb = cseq.BwdAdjacency;
1926
1927            % find state candidates and their rankings
1928            [S, Rs] = knet.findStates(data(ti), gmm, verbose);
1929
1930            % downselect to only N states
1931            if ¬isempty(N)
1932                S = S(1:N);
1933            end
1934
1935            % search through transition matrix for the actions
1936            [A, Pa] = knet.findActions(Wa, Wb, S, verbose);
1937
1938            % continue to evaluate the model until the stat
1939            counter = 1;
1940            done = false;
1941            while ¬done
1942
```

```
1943            % find the action sequence
1944            aseq = knet.actionSequence(data(ti), gmm, A, S, 0);
1945
1946            % learn the action sequence
1947            trans = knet.learnSequence(M, aseq, A, S, 1, [], verbose, 0);
1948
1949            % get the Likelihood estimates (for the training indices)
1950            [logLikT, ns] = knet.likelihood(data(ti), gmm, trans,...
1951                A, S, includeSeq);
1952
1953            % get the likelihood estimates (for the validation indices)
1954            [logLikV, ¬] = knet.likelihood(data(vi), gmm, trans,...
1955                A, S, includeSeq);
1956
1957            % get the model covariance
1958            [Ra, Rf] = knet.modelUncertainty(data(ti), gmm, A, aseq);
1959
1960            % compute the number of parameters
1961            Dim = gmm.NumVariables;
1962            np = length(A)*gmm.NumComponents +...
1963                 length(S) +...
1964                 trans.net.numWeightElements +...
1965                 gmm.NumComponents*(Dim^2 + Dim);
1966
1967            % save the model strcture
1968            model{counter}.LogLikelihoodTraining = logLikT;
1969            model{counter}.LogLikelihoodValidation = logLikV;
1970            model{counter}.FwdAdjacency = Wa;
1971            model{counter}.BwdAdjacency = Wb;
```

```
1972                model{counter}.GMM = gmm;
1973                model{counter}.Transition = trans;
1974                model{counter}.States = S;
1975                model{counter}.Actions = A;
1976                model{counter}.Ra = Ra;
1977                model{counter}.Rf = Rf;
1978                model{counter}.NumSamples = ns;
1979                model{counter}.NumParameters = np;
1980                model{counter}.BIC = log(ns).*np - 2*logLikT;
1981                model{counter}.AIC = 2*np - 2*logLikT;
1982                model{counter}.AICc = 2*np - 2*logLikT +...
1983                    (2*np^2 + 2*np)./(ns - np -1);
1984
1985                % check end condition and iterate
1986                if ¬isempty(N)
1987
1988                    % move on
1989                    done = true;
1990
1991                else
1992                    % remove the weakest S
1993                    S = S(1:end-1);
1994
1995                    % find actions
1996                    [A, Pa] = knet.findActions(Wa, Wb, S, verbose);
1997
1998                    % if A is empty, we're done
1999                    done = isempty(A);
2000
```

```
2001                % iterate the model counter
2002                counter = counter + 1;
2003            end
2004        end
2005    end
2006    end
2007
2008
2009    % Optimization functions
2010    methods (Static)
2011
2012        function J = costTrajectory(x, action, gmm, ch)
2013
2014            % reshape data
2015            xi = [ch.x ch.v];
2016            d = length(xi);
2017            dt = x(end);
2018            xv = x(1:end-1);
2019            xv = reshape(xv, [d, length(xv)/d]);
2020
2021            % integrate to compute the next step
2022            xvHat = knet.rk4singlestep(...
2023                @(t,x)knet.predictMotion(t, x, action, gmm, ch),...
2024                dt,0,xv(:,1:end-1)')';
2025
2026            % compute vector projection
2027            J = sum(sum((xvHat - xv(:,2:end)).^2));
2028        end
2029
```

```
2030        function [c,ceq] = constraintTrajectory(x, gmm, indices, ch)
2031
2032            % reshape data
2033            xi = [ch.x ch.v];
2034            d = length(xi);
2035            dt = x(end);
2036            xv = x(1:end-1);
2037            xv = reshape(xv, [d, length(xv)/d]);
2038
2039            % compute error on ends
2040            Mu1 = xv(:,1)' - gmm.mu(indices(1),xi);
2041            Sig1 = gmm.Sigma(xi,xi,indices(1));
2042            Mu2 = xv(:,end)' - gmm.mu(indices(2),xi);
2043            Sig2 = gmm.Sigma(xi,xi,indices(2));
2044            ceq = Mu1 * inv(Sig1) * Mu1' +...
2045                Mu2 * inv(Sig2) * Mu2';
2046            c = 0;
2047        end
2048    end
2049
2050
2051    % Probability helpers
2052    methods (Static)
2053
2054        function [lik, arg, gain] = gaussian(x, mu, sigma)
2055            % [lik, arg, gain] = gaussian(x, mu, sigma)
2056            %
2057            % This function return elements of multivariate normal gaussian
2058            % distribution.
```

```
2059            %
2060            % lik = exp(-arg / 2)
2061            % arg = (x-\mu)' inv(Sigma) (x-\mu)
2062            % gain = 1 / sqrt( \det(Sigma) * (2 \pi)^d )
2063            %
2064            % x      - test states (d x n)
2065            % mu     - mean state (d x 1) or (d x n)
2066            % sigma  - covariance matrix (d x d) or (d x d x n)
2067            % lik    - the gaussian likelihood
2068            % arg    - the mahalanobis distance to the mean
2069            % gain   - the multiplier such that int(x) = 1
2070            e = bsxfun(@minus, x, mu);
2071            [d, n] = size(x);
2072            ns = size(sigma,3);
2073            if ns ≠1
2074                ds = zeros(1, ns);
2075                a = zeros(size(e))';
2076                L = zeros(size(sigma));
2077                for k = 1:ns
2078                    L(:,:,k) = chol(inv(sigma(:,:,k)),'lower');
2079                    a(k,:) = e(:,k)' * L(:,:,k);
2080                    ds(k) = det(sigma(:,:,k));
2081                end
2082            else
2083                L = chol(inv(sigma),'lower');
2084                a = e' * L;
2085                ds = det(sigma);
2086            end
2087            arg = sum(a.^2,2)';
```

```
2088            lik = exp(-arg ./ 2);
2089            gain = 1 ./ sqrt( ds * (2 * pi).^d );
2090        end
2091    end
2092
2093
2094    % Integration
2095    methods (Static)
2096
2097        function y1 = rk4singlestep(fun,dt,t0,y0)
2098            % y1 = rk4singlestep(fun,dt,t0,y0)
2099            %
2100            % This function performs a single step rk4 integration scheme.
2101            %
2102            % fun  - handle to the function @(t,x)
2103            % dt   - time step
2104            % t0   - time (current)
2105            % y0   - state (current)
2106            % y1   - state (next)
2107            f1 = fun(t0,y0);
2108            f2 = fun(t0+dt/2,y0+(dt/2)*f1);
2109            f3 = fun(t0+dt/2,y0+(dt/2)*f2);
2110            f4 = fun(t0+dt,y0+dt*f3);
2111            y1 = y0 + (dt/6)*(f1+2*f2+2*f3+f4);
2112        end
2113
2114        function y1 = fesinglestep(fun,dt,t0,y0)
2115            % y1 = fesinglestep(fun,dt,t0,y0)
2116            %
```

```
2117            % This function performs a single step forward euler
2118            % integration scheme.
2119            %
2120            % fun  - handle to the function @(t,x)
2121            % dt   - time step
2122            % t0   - time (current)
2123            % y0   - state (current)
2124            % y1   - state (next)
2125            f1 = fun(t0,y0);
2126            y1 = y0 + dt*f1;
2127        end
2128    end
2129
2130
2131    % Data helpers
2132    methods (Static)
2133
2134        function data = emptyDataStructure()
2135            % data = getDataStructure()
2136            %
2137            % This function creates an empty data structure for the
2138            % variables needed for training, validation, and test.
2139            %
2140            % data - the empty data structure
2141            data.meas.X = []; % unfiltered position data
2142            data.meas.V = []; % unfiltered velocity data
2143            data.meas.A = []; % unfiltered acceleration data
2144            data.meas.F = []; % unfiltered force data
2145            data.X = []; % position data
```

```
2146            data.V = []; % velocity data
2147            data.A = []; % acceleration data
2148            data.F = []; % force data
2149            data.T = []; % time
2150            data.id_cluster = []; % user-supplied cluster data indices
2151            data.name = []; % file name
2152        end
2153
2154        function [D, ch] = processData(data)
2155            % [D, ch] = processData(data)
2156            %
2157            % This function processes the data structure and returns a
2158            % compiled matrix D and channel ch indices.
2159            %
2160            % data - the populated data structure
2161            % D    - the data matrix (rows: observations, cols: signals)
2162            % ch   - indices indicating which cols of D are which signals
2163            D = [data.X, data.V, data.A, data.F, data.T];
2164            ch.x = 1:size(data.X,2);
2165            ch.v = ch.x(end)+(1:size(data.V,2));
2166            ch.a = ch.v(end)+(1:size(data.A,2));
2167            ch.f = ch.a(end)+(1:size(data.F,2));
2168            ch.t = ch.f(end)+(1:size(data.T,2));
2169        end
2170
2171        function [idt, idv] = splitTrainingData(trPct, n)
2172            % [idt, idv] = splitTrainingData(trPct, n)
2173            %
2174            % This function randomly splits the data into training and
```

```
2175            % validation, preserving the order of the original sequence.
2176            %
2177            % trPct   - ratio of data to use for training [0-1]
2178            % n       - number of points
2179            % idt     - indices of the training data
2180            % idv     - indices of the validation data
2181            id_p = randperm(n);
2182            N = round(trPct * n);
2183            idt = sort(id_p(1:N));
2184            idv = sort(id_p(N+1:end));
2185        end
2186
2187        function id = randomizeTrainingData(Nsamples, n)
2188            % id = randomizeTrainingData(Nsamples, n)
2189            %
2190            % This function randomly downselects the data from n samples
2191            % into Nsamples
2192            %
2193            % Nsamples - number of samples to downselect to
2194            % n       - number of original samples
2195            % id      - indices of the training data
2196            N = min(Nsamples,n);
2197            rp = randperm(n)';
2198            id = rp(1:N);
2199        end
2200    end
2201
2202
2203    % Plotting
```

```
2204    methods (Static)
2205
2206        function optn = plotOptions()
2207            % optn = plotOptions()
2208            %
2209            % This function returns a structure of plot options
2210            %
2211            % optn - structure of plot option settings
2212
2213            % Color palette
2214            clr.yellow = [.9 .9 .1]; % [1 .8 .1]
2215            clr.yellowGray = [.7 .7 .5];
2216            clr.red = [.8 .1 .1];
2217            clr.redGray = [.7 .5 .5];
2218            clr.green = [.1 .8 .1];
2219            clr.greenGray = [.5 .7 .5];
2220            clr.blueNavy = [.1 .1 .42];
2221            clr.blue = [.1 .1 .8];
2222            clr.blueGray = [.5 .5 .7];
2223            clr.white = [.9 .9 .9];
2224            clr.grayLight = [.7 .7 .7];
2225            clr.gray = [.5 .5 .5];
2226            clr.grayDark = [.3 .3 .3];
2227            clr.black = [.06 .05 .05];
2228
2229            % Shared options
2230
2231            % channels
2232            optn.channel = 1;
```

```
2233            optn.action = [];
2234
2235            % display units
2236            optn.units.Position = '(m)';
2237            optn.units.Velocity = '(m/s)';
2238            optn.units.Accel = '(m/s/s)';
2239            optn.units.Effort = '(N)';
2240
2241             % display range (sets axis limits)
2242            optn.range.Position = [];
2243            optn.range.Velocity = [];
2244            optn.range.Accel = [];
2245            optn.range.Effort = [];
2246
2247            % define scale factors (for vector plots)
2248            optn.scale.Velocity = 0.02;
2249            optn.scale.Effort = 0.02;
2250            optn.scale.Accel = 0.02;
2251
2252            % default color/transparency options
2253            optn.color.Data = clr.grayDark;
2254            optn.color.AccelVectorCluster = clr.red;
2255            optn.color.AccelVectorTraining = clr.redGray;
2256            optn.color.EffortVectorCluster = clr.yellow;
2257            optn.color.EffortVectorTraining = clr.yellowGray;
2258            optn.color.MuInnerClusters = clr.blue;
2259            optn.color.MuOuterClusters = clr.red;
2260            optn.color.MuInnerTargets = clr.yellow;
2261            optn.color.MuOuterTargets = clr.red;
```

```
2262            optn.color.HmmSequence = clr.blueNavy;
2263            optn.color.ClusterText = [1 1 1];
2264            optn.color.SigmaClusters = clr.blue;
2265            optn.alpha.SigmaClusters = 0.2;
2266            optn.conf.Interval = 0.9545; % 2 sigma
2267
2268            % size options
2269            optn.size.Data = 1;
2270            optn.size.MuClusters = 8;
2271            optn.size.MuTargets = 10;
2272            optn.size.HmmSequence = 8;
2273            optn.size.ClusterVector = 4;
2274            optn.size.ClusterText = 16;
2275            optn.size.DataClustered = 2;
2276
2277            % define default display settings
2278            optn.show.Title = false;
2279            optn.show.Data = true;
2280            optn.show.EffortVectorTraining = false;
2281            optn.show.AccelVectorTraining = false;
2282            optn.show.EffortVectorCluster = false;
2283            optn.show.AccelVectorCluster = false;
2284            optn.show.CentersClusters = true;
2285            optn.show.CentersTargets = true;
2286            optn.show.Sequence = false;
2287            optn.show.CovarianceClusters = true;
2288            optn.show.ClusterNames = true;
2289            optn.show.ClusteredData = false;
2290
```

```
2291            % Vector diagram
2292            optn.vectorDiagram.number = 1;
2293
2294            % State connection diagram
2295            optn.connectionDiagram.number = 2;
2296
2297            % Action sequence plot
2298            optn.actionSequence.number = 3;
2299
2300            % Likelihood simulation options
2301            optn.likAnimation.number = [];
2302
2303            % Color pallete
2304            optn.palette = clr;
2305        end
2306
2307        function h = plotStateConnection(A, States, S0, Sf, optn)
2308            % h = plotStateConnection(sequence, showTargets, optn)
2309            %
2310            % This function plots the state connection diagram from the
2311            % sequence object
2312            %
2313            % A          - stochastic matrix
2314            % States     - vector of state indices
2315            % S0         - starting state
2316            % Sf         - ending state
2317            % findNumber - number to the figure
2318            % h          - figure handle
2319            h = figure(optn.connectionDiagram.number); clf
```

```
2320            h=colordef(h,'white'); %Set color scheme
2321            h.Color='w'; %Set background color of figure window
2322            hold on; axis off;
2323
2324            % set plot settings
2325            Rad = 1;
2326            theta = linspace(0,2*pi,length(A)+1);
2327            theta = theta(1:end-1);
2328            MaxWidth = 12;
2329            MaxLength = 20;
2330
2331            % plot all the states
2332            scatter(Rad*cos(theta),Rad*sin(theta),10,...
2333                'MarkerEdgeColor',[.1 .1 .25],...
2334                'MarkerFaceColor',[.9 .9 .9])
2335            axis equal
2336            for i = 1:length(A)
2337                for j = 1:length(A)
2338                    start = Rad*[cos(theta(j)) sin(theta(j))];
2339                    stop = Rad*[cos(theta(i)) sin(theta(i))];
2340                    p = (A(i,j) + A(j,i))/2;
2341                    wi = MaxWidth * p;
2342                    % if it happens more than once, believe it
2343                    if A(i,j) > eps
2344                        arrow(start, stop, 'Width', wi,'Length',MaxLength,...
2345                            'FaceColor',optn.color.HmmSequence,...
2346                            'EdgeColor',optn.color.HmmSequence)
2347                    end
2348                end
```

```
2349            end
2350            scatter(Rad*cos(theta),Rad*sin(theta),600,...
2351                'MarkerEdgeColor',[.1 .1 .25],...
2352                'MarkerFaceColor',[.9 .9 .9])
2353            scatter(Rad*cos(theta(States)),...
2354                Rad*sin(theta(States)),600,...
2355                'MarkerEdgeColor',[.1 .1 .25],...
2356                'MarkerFaceColor',[.9 .9 .1])
2357            scatter(Rad*cos(theta(S0)),...
2358                Rad*sin(theta(S0)),600,...
2359                'MarkerEdgeColor',[.1 .1 .25],...
2360                'MarkerFaceColor',[.2 1 .3])
2361            scatter(Rad*cos(theta(Sf)),...
2362                Rad*sin(theta(Sf)),600,...
2363                'MarkerEdgeColor',[.1 .1 .25],...
2364                'MarkerFaceColor',[1 .2 .3])
2365            for i = 1:length(A)
2366                text(Rad*cos(theta(i)), Rad*sin(theta(i)),...
2367                    sprintf('$c_{%i}$',i),'HorizontalAlignment','center',...
2368                    'VerticalAlignment','middle','Interpreter','latex')
2369            end
2370            drawnow
2371        end
2372
2373        function h = plot(data, gmm, Wa, S, A, optn)
2374            % h = plot(data, gmm, Wa, S, A, optn)
2375            %
2376            % This function plots the data with the clusters and sequence
2377            % information overlayed
```

```
2378            %
2379            % data     - data structure
2380            % gmm      - clustered data structure
2381            % Wa       - the adjacency transition matrix
2382            % S        - vector of states
2383            % A        - set of action manifolds
2384            % figNumber - figure number
2385            % h        - figure handle
2386            % generate default plot options
2387            if nargin < 6
2388                optn = knet.plotOptions();
2389            end
2390
2391            % create figure
2392            h = figure(optn.vectorDiagram.number); clf
2393            h=colordef(h,'white'); %Set color scheme
2394            h.Color='w'; %Set background color of figure window
2395            hold on;
2396
2397            % set the axis limits
2398            if ¬isempty(optn.range.Position)
2399                xlim(optn.range.Position)
2400            end
2401            if ¬isempty(optn.range.Velocity)
2402                ylim(optn.range.Velocity)
2403            end
2404
2405            % Initialize the plots
2406            xlabel(['Position ' optn.units.Position ''])
```

```
2407            ylabel(['Velocity ' optn.units.Velocity ''])
2408
2409            % get the channels
2410            [¬,ch] = knet.processData(data(1));
2411            xi = [ch.x(optn.channel) ch.v(optn.channel)];
2412
2413            % show the action manifold
2414            if (¬isempty(optn.action) && ¬isempty(A))
2415                xl = xlim;
2416                vl = ylim;
2417                x = linspace(xl(1),xl(end),31);
2418                v = linspace(vl(1),vl(end),31);
2419                [xm,vm] = meshgrid(x,v);
2420
2421                in = [ch.x ch.v];
2422                Xp = zeros(length(xm(:)),length(in));
2423                Xp(:,ch.x(optn.channel)) = xm(:);
2424                Xp(:,ch.v(optn.channel)) = vm(:);
2425                Xpdot = knet.predictMotion(0, Xp,...
2426                    A{optn.action}, gmm, ch);
2427
2428                quiver(xm(:), vm(:),...
2429                    optn.scale.Velocity * Xpdot(:,ch.x(1)),...
2430                    optn.scale.Accel * Xpdot(:,ch.v(1)),...
2431                    'Color',optn.color.AccelVectorCluster,...
2432                    'ShowArrowHead','off','LineWidth',1);
2433            end
2434
2435            % plot the data cluster covariances
```

```
2436            if (optn.show.CovarianceClusters) && ¬isempty(gmm)
2437                n = length(gmm.ComponentProportion);
2438                for i = 1:n
2439                    if strcmp(gmm.CovarianceType,'diagonal')
2440                        Sigma = diag(gmm.Sigma(1,xi,i));
2441                    else
2442                        Sigma = gmm.Sigma(xi,xi,i);
2443                    end
2444                    he=error_ellipse(Sigma,...
2445                        gmm.mu(i,xi),...
2446                        'conf',optn.conf.Interval);
2447                    patch('XData',he.XData,'YData',he.YData,...
2448                        'FaceColor',optn.color.SigmaClusters,...
2449                        'FaceAlpha',optn.alpha.SigmaClusters,...
2450                        'EdgeColor','none');
2451                    delete(he)
2452                end
2453            end
2454
2455            % plot the data
2456            if (optn.show.Data) && ¬isempty(data)
2457                for k = 1:length(data)
2458                    plot(data(k).X(:,optn.channel),...
2459                         data(k).V(:,optn.channel), '-',...
2460                         'Color', optn.color.Data,...
2461                         'Linewidth', optn.size.Data)
2462                end
2463            end
2464
```

```
2465            % plot the data vector
2466            for k = 1:length(data)
2467                id = data(k).id_cluster;
2468                if (optn.show.EffortVectorTraining) && ¬isempty(data)
2469                    quiver(data(k).X(id,optn.channel),...
2470                        data(k).V(id,optn.channel),...
2471                        zeros(size(data(k).V(id,optn.channel))),...
2472                        optn.scale.Effort*data(k).F(id,optn.channel),0,...
2473                        'Color',optn.color.EffortVectorTraining,...
2474                        'ShowArrowHead','off','LineWidth',1);
2475                end
2476                if (optn.show.AccelVectorTraining) && ¬isempty(data)
2477                    quiver(data(k).X(id,optn.channel),...
2478                        data(k).V(id,optn.channel),...
2479                        optn.scale.Velocity*data(k).V(id,optn.channel),...
2480                        optn.scale.Accel*data(k).A(id,optn.channel),0,...
2481                        'Color',optn.color.AccelVectorTraining,...
2482                        'ShowArrowHead','off','LineWidth',1);
2483                end
2484                if (optn.show.ClusteredData) && ¬isempty(gmm)
2485                    [D, ch] = knet.processData(data(k));
2486                    vars = [ch.x ch.v ch.a ch.f];
2487                    id = data(k).id_cluster;
2488                    C = gmm.cluster(D(id,vars));
2489                    scatter(data(k).X(id,optn.channel),...
2490                           data(k).V(id,optn.channel),...
2491                           optn.size.DataClustered,C,'filled');
2492                end
2493            end
```

```
2494
2495            % plot the hmm sequence
2496            if (optn.show.Sequence && ¬isempty(Wa))
2497                for i = 1:gmm.NumComponents
2498                    for j = 1:gmm.NumComponents
2499                        wi = optn.size.HmmSequence * Wa(i,j);
2500                        if Wa(i,j) > eps
2501                            plot(gmm.mu([i j],xi(1)),gmm.mu([i j],xi(2)),...
2502                                'Linewidth', wi,...
2503                                'Color',optn.color.HmmSequence)
2504                        end
2505                    end
2506                end
2507            end
2508
2509            % plot the cluster vector
2510            if (optn.show.EffortVectorCluster) && ¬isempty(gmm)
2511                quiver(gmm.mu(:,ch.x(optn.channel)),...
2512                    gmm.mu(:,ch.v(optn.channel)),...
2513                    zeros(size(gmm.mu(:,ch.v(optn.channel)))),...
2514                    optn.scale.Effort*gmm.mu(:,ch.f(optn.channel)),0,...
2515                    'Color',optn.color.EffortVectorCluster,...
2516                    'ShowArrowHead','off',...
2517                    'LineWidth',optn.size.ClusterVector);
2518            end
2519            if (optn.show.AccelVectorCluster) && ¬isempty(gmm)
2520                quiver(gmm.mu(:,ch.x(optn.channel)),...
2521                    gmm.mu(:,ch.v(optn.channel)),...
2522                    optn.scale.Velocity*gmm.mu(:,ch.v(optn.channel)),...
```

```
2523                optn.scale.Accel*gmm.mu(:,ch.a(optn.channel)),0,...
2524                'Color',optn.color.AccelVectorCluster,...
2525                'ShowArrowHead','off',...
2526                'LineWidth',optn.size.ClusterVector);
2527            end
2528
2529            % plot the data cluster centers
2530            if (optn.show.CentersClusters) && ¬isempty(gmm)
2531                plot(gmm.mu(:,xi(1)),...
2532                     gmm.mu(:,xi(2)),'o',...
2533                     'MarkerEdgeColor',optn.color.MuOuterClusters,...
2534                     'MarkerFaceColor',optn.color.MuInnerClusters,...
2535                     'MarkerSize',optn.size.MuClusters)
2536            end
2537
2538            % plot the data target centers
2539            if (optn.show.CentersTargets && ¬isempty(S))
2540                plot(gmm.mu(S,xi(1)),...
2541                     gmm.mu(S,xi(2)),'o',...
2542                     'MarkerEdgeColor',optn.color.MuOuterTargets,...
2543                     'MarkerFaceColor',optn.color.MuInnerTargets,...
2544                     'MarkerSize',optn.size.MuTargets)
2545            end
2546
2547            % show the cluster names in text
2548            if (optn.show.ClusterNames) && ¬isempty(gmm)
2549                for k=1:gmm.NumComponents
2550                    text(gmm.mu(k,xi(1)),gmm.mu(k,xi(2)),...
2551                         sprintf('$c_{%i}$',k),'Interpreter','latex',...
```

```
                        'Color',optn.color.ClusterText,...
                        'FontSize',optn.size.ClusterText)
                end
            end

            % draw the title
            if optn.show.Title && ¬isempty(gmm)
                str = sprintf('K = %i',gmm.NumComponents);
                suptitle(str)
            end
            drawnow
        end
    end
end
```

C.1.2 Training script

Listing C.2: matlab/learnTask.m - script for learning the policy in the experiments using knet.m

```
% Copyright 2014-2019, Parker Owan
% PhD Dissertation, University of Washington

addpath GMM-GMR-v2.0
clear all

% Font size (mac: 16, Windows: 12)
FontSize = 16;

```

```
% Set the interpretter
warning off
set(0,'DefaultTextFontSize',FontSize,...
      'DefaultAxesFontSize',FontSize,...
      'DefaultTextInterpreter','Latex',...
      'DefaultLegendInterpreter','Latex',...
      'DefaultLineLineWidth',1,...
      'DefaultLineMarkerSize',7.75)


% ************************************************
%% OPTIONS
% ************************************************

% verbose option (print out results)
verbose = true;

% show preprocess figures
showProcFigures = 0;

% choose the dataset to use
dataset = 1;      % 1 : kinesthetic (new robot)
                  % 2 : haptic (new robot)
                  % 3 : kinesthetic (old robot)
                  % 4 : combined 1 and 2

% position of interest ('handle' or 'tip')
poi = 'handle';
```

```
39
40 % write the model to memory
41 saveModel = 0;
42
43 % data collection sample rate (s) (data gets interpolated to this)
44 dt = 1/1000;
45
46 % training sample rate (s) training data is downsampled to this)
47 % dtTrain = dt * 50; % needs to be integer multiple of dt
48 dtTrain = dt * 5; % needs to be integer multiple of dt
49
50 % mask for active (non-singular variables)
51 activeChannels = boolean([1 1 0 0 0 0]);
52
53 % cluster mode
54 clusterMode = 'full'; % 'diagonal' or 'full'
55
56 % range of number of clusters to evaluate
57 % Note: upper bound should be chosen such that Nparams << NSamples
58 % K = 2:14;
59 K = 8;
60
61 % Number of neural network layers
62 Nlayers = 20;
63
64 % number of states (leave empty if we don't know)
65 % Nstates = [];
66 Nstates = 3;
67
```

```
% include the forward-chained sequence in the likelihood evaluation
includeSeq = true;

% select the number (p) of terms for leave-p-out cross validation. If this
% parameter is zero to zero, then cross-validation will not be performed.
% Note that cross-validation requires significantly longer training times.
np = 0;

% data pre-filter options (fstop should be 1/10 dtTrain)
fcutoff = 10;    % Hz (cutoff freq)
fstop = 30;      % Hz (stopband freq)

% seed the random number generator (for consistency)
rng(3)


% ************************************************
%% Select the data
% ************************************************

switch dataset
    case 1
        % Kinesthetic dataset (new robot)
        files = {'data/test_s00_t100_r000'
                'data/test_s00_t100_r001' % stops
                'data/test_s00_t100_r002'
                'data/test_s00_t100_r003'
                'data/test_s00_t100_r004'
                'data/test_s00_t100_r005'
```

```
97              'data/test_s00_t100_r006'
98              'data/test_s00_t100_r007'
99              'data/test_s00_t100_r008'
100             'data/test_s00_t100_r009' % comes out of hole
101             'data/test_s00_t100_r010' % stops
102             'data/test_s00_t100_r011'
103             'data/test_s00_t100_r012'
104             'data/test_s00_t100_r013'
105             'data/test_s00_t100_r014'};
106     case 2
107         % Haptic dataset (new robot)
108         files = {'data/test_s00_t101_r000' %
109                 'data/test_s00_t101_r001'
110                 'data/test_s00_t101_r002'
111                 'data/test_s00_t101_r003'
112                 'data/test_s00_t101_r004'
113                 'data/test_s00_t101_r005'
114                 'data/test_s00_t101_r006'
115                 'data/test_s00_t101_r007' %
116                 'data/test_s00_t101_r008'
117                 'data/test_s00_t101_r009' %
118                 'data/test_s00_t101_r010' %
119                 'data/test_s00_t101_r011'
120                 'data/test_s00_t101_r012' %
121                 'data/test_s00_t101_r013'
122                 'data/test_s00_t101_r014'};
123     case 3
124         % Kinesthetic dataset (old robot)
125         files = {'data/test_s00_t000_r027'
```

```
126                 'data/test_s00_t000_r028'
127                 'data/test_s00_t000_r029'
128                 'data/test_s00_t000_r030'
129                 'data/test_s00_t000_r031'
130                 'data/test_s00_t000_r032'
131                 'data/test_s00_t000_r033'
132                 'data/test_s00_t000_r035'};
133
134     case 4
135         % combined 1 and 2
136         files = {'data/test_s00_t100_r000'
137                 'data/test_s00_t100_r001' % stops
138                 'data/test_s00_t100_r002'
139                 'data/test_s00_t100_r003'
140                 'data/test_s00_t100_r004'
141                 'data/test_s00_t100_r005'
142                 'data/test_s00_t100_r006'
143                 'data/test_s00_t100_r007'
144                 'data/test_s00_t100_r008'
145                 'data/test_s00_t100_r009' % comes out of hole
146                 'data/test_s00_t100_r010' % stops
147                 'data/test_s00_t100_r011'
148                 'data/test_s00_t100_r012'
149                 'data/test_s00_t100_r013'
150                 'data/test_s00_t100_r014'
151                 'data/test_s00_t101_r000' %
152                 'data/test_s00_t101_r001'
153                 'data/test_s00_t101_r002'
154                 'data/test_s00_t101_r003'
```

```
155                 'data/test_s00_t101_r004'
156                 'data/test_s00_t101_r005'
157                 'data/test_s00_t101_r006'
158                 'data/test_s00_t101_r007' %
159                 'data/test_s00_t101_r008'
160                 'data/test_s00_t101_r009' %
161                 'data/test_s00_t101_r010' %
162                 'data/test_s00_t101_r011'
163                 'data/test_s00_t101_r012' %
164                 'data/test_s00_t101_r013'
165                 'data/test_s00_t101_r014'};
166 end
167
168 % ************************************************
169 %% DATA-PROCESSING
170 % ************************************************
171
172 % post-process
173 data = postProcessData(files, showProcFigures, fcutoff, fstop, dt,...
174     dtTrain, poi, activeChannels);
175
176 % get batch information
177 Nsamples = 0;
178 for k = 1:length(data)
179     Nsamples = Nsamples + length(data(k).id_cluster);
180 end
181
182 % leave-p-out cross-validation
183 dval = nchoosek(1:length(data),np);
```

```
184
185 % start by centering the data
186 [data, center] = knet.centerData(data);
187
188 % compute the measuremenet noise
189 noise = knet.measurementNoise(data);
190
191
192 % ************************************************
193 %% LEARN THE KINETIC NET
194 % ************************************************
195
196 fprintf('************************************************\n')
197 fprintf('STARTING K-KNET\n');
198 fprintf('Copyright (2018) Parker Owan\n')
199 fprintf('************************************************\n')
200 fprintf('N = %i samples selected for training.\n',Nsamples);
201
202 % get the channels
203 [¬,ch] = knet.processData(data(1));
204
205 % for each number of clusters
206 fprintf('Beginning clustering sequence with Kmax = %i.\n',K(end));
207 model = {};
208 xval = {};
209 for k = 1:length(K)
210
211     % perform cross-validation round
212     if np > 0
```

```
213         for i = 1:length(dval)
214             vi = dval(i,:); % validation indices
215             ti = 1:length(data); % training indices
216             ti(find(sum(vi == ti',2))) = []; % remove val from train
217             batch = knet.train(data, ti, vi, K(k), Nstates, Nlayers,...
218                 clusterMode, includeSeq, verbose);
219             xval{k,i} = batch;
220         end
221     end
222
223     % train several models
224     ti = 1:length(data); % training indices
225     batch = knet.train(data, ti, [], K(k), Nstates, Nlayers,...
226         clusterMode, includeSeq, verbose);
227
228     % append the model
229     model = [model, batch];
230 end
231
232
233 % ************************************************
234 %% Model selection
235 % ************************************************
236
237 % clean up the xvalidation
238 counter = 1;
239 for i = 1:size(xval,1)
240     for j = 1:i
241         LogLikVal(counter) = 0;
```

```
242         for k = 1:size(xval,2)
243             LogLikVal(counter) = LogLikVal(counter) +...
244                 xval{i,k}{j}.LogLikelihoodValidation;
245         end
246         counter = counter + 1;
247     end
248 end
249 
250 % get the parameters from the model structure
251 LogLikFull = zeros(length(data),length(model));
252 AICFull = zeros(length(data),length(model));
253 BICFull = zeros(length(data),length(model));
254 NParamFull = zeros(length(data),length(model));
255 NSamplesFull = zeros(length(data),length(model));
256 ParamS3 = zeros(length(data),length(model));
257 for i = 1:length(model)
258     NParam(i) = model{i}.NumParameters;
259 
260     % get the expected values of everything
261     ns = mean(model{i}.NumSamples);
262     LogLik(i) = mean(model{i}.LogLikelihoodTraining);
263     AIC(i) = -2*LogLik(i) + 2*model{i}.NumParameters;
264     BIC(i) = -2*LogLik(i) + model{i}.NumParameters*log(ns);
265 
266     % get the full set of values of everything
267     NParamFull(:,i) = model{i}.NumParameters;
268     NSamplesFull(:,i) = model{i}.NumSamples;
269     LogLikFull(:,i) = model{i}.LogLikelihoodTraining';
270     AICFull(:,i) = model{i}.AIC';
```

```
271     BICFull(:,i) = model{i}.BIC';
272
273     % get the full set of values of everything
274     if length(model{i}.States)==3
275         ParamS3(:,i) = 1;
276     end
277 end
278
279 % find the mins
280 ilik = find(min(-LogLik) == -LogLik,1,'first');
281 ibic = find(min(BIC) == BIC,1,'first');
282 iaic = find(min(AIC) == AIC,1,'first');
283
284 % plot the parameter info
285 figure(100),clf
286 hold on; grid on;
287 scatter(NParam,-2*LogLik,'filled','Marker','o','MarkerFaceAlpha',0.6,...
288     'MarkerFaceColor','k')
289 scatter(NParam,AIC,'filled','Marker','o','MarkerFaceAlpha',0.6)
290 scatter(NParam,BIC,'filled','Marker','o','MarkerFaceAlpha',0.6)
291 plot(NParam(ilik),-2*LogLik(ilik),'sb','MarkerSize',15)
292 plot(NParam(iaic),AIC(iaic),'+b','MarkerSize',15)
293 plot(NParam(ibic),BIC(ibic),'xb','MarkerSize',15)
294 legend('$-2\ln(L)$','AIC','BIC','min $-2\ln(L)$','min AIC','min BIC',...
295     'Location','NorthWest')
296 ylabel('Performance')
297 xlabel('Number of parameters')
298 set(gcf,'position',[440 454 560 300])
299
```

```
300
301 % plot the parameter info
302 figure(101),clf
303 hold on; grid on;
304 scatter(NParamFull(:),-2*LogLikFull(:)./NSamplesFull(:),'filled',...
305     'Marker','o','MarkerFaceAlpha',0.6,'MarkerFaceColor','k')
306 scatter(NParamFull(:),AICFull(:)./NSamplesFull(:),'filled','Marker','o',...
307     'MarkerFaceAlpha',0.6)
308 legend('$-2\ln(L)$','AIC',...
309     'Location','NorthWest')
310 ylabel('Performance')
311 xlabel('Number of parameters')
312 set(gcf,'position',[440 454 560 300])
313
314
315 % plot the parameter info
316 id = find(ParamS3 > 0);
317 figure(102),clf
318 hold on; grid on;
319 scatter(NParamFull(id),-2*LogLikFull(id)./NSamplesFull(id),'filled',...
320     'Marker','o','MarkerFaceAlpha',0.6,'MarkerFaceColor','k')
321 scatter(NParamFull(id),AICFull(id)./NSamplesFull(id),'filled',...
322     'Marker','o','MarkerFaceAlpha',0.6)
323 legend('$-2\ln(L)$','AIC',...
324     'Location','NorthWest')
325 ylabel('Performance')
326 xlabel('Number of parameters')
327 set(gcf,'position',[440 454 560 300])
328 ylim([-18 -4])
```

```
329 xlim([0 2500])
330
331 % extract the selected model
332 mdl = model{iaic};
333
334 % get the state-space models
335 [A, B, C, D, Q, R] = knet.stateSpace(mdl.GMM, ch);
336
337
338 % ************************************************
339 %% Learn the trajectories for the selected model
340 % ************************************************
341
342 % time step for sampling the initial trajectory for the optimizers
343 dtTraj = 1/50; % (s)
344
345 % learn each trajectory
346 traj = knet.learnActionTrajectories(mdl.GMM, mdl.Actions, ch, dtTraj);
347
348 % init the brush kinetic net object
349 BrushKnet = knet();
350
351 % populate with the parameters
352 BrushKnet.populate(mdl, traj, ch, center, noise);
353
354 % plot the learned trajectories
355 optn = knet.plotOptions;
356 for k = 1:length(BrushKnet.mTrajectories)
357     optn.action = k;
```

```
358     optn.range.Position = [-0.05 0.05];
359     optn.range.Velocity = [-0.40 0.40];
360     optn.vectorDiagram.number = k;
361     h = knet.plot(data, BrushKnet.mGmm,...
362         [], BrushKnet.mStates, BrushKnet.mActions, optn);
363     plot(BrushKnet.mTrajectories(k).x(1,:),...
364         BrushKnet.mTrajectories(k).v(1,:),'-r','LineWidth',4);
365     set(h,'position',[90 210 560 300]);
366 end
367
368
369 % ************************************************
370 %% Write the model to a 'Knet' object
371 % ************************************************
372
373 % save the BrushNet object
374 if (saveModel)
375     filename = ['knet_' num2str(now),'.mat'];
376     save(filename,'BrushKnet');
377 end
```

C.1.3 Data collection scripts

Listing C.3: matlab/test_demonstration.m - script for collecting demonstration data on hardware

```
1 % Copyright 2014-2019, Parker Owan
2 % PhD Dissertation, University of Washington
3 %
```

```
4  % Script for collecting demonstration data (haptically or kinesthetically)
5
6  clear all
7  clc
8
9  % add paths
10 addpath hebi
11 warning off
12
13 % --------------------------------------
14 %% OPTIONS
15 % --------------------------------------
16
17 % number of demonstrations
18 Nd = 4;
19
20 % countdown intervals (1 = 1 s)
21 Nc = 4;
22
23 % minimum time passed before test can end
24 tmin = 1; % (s)
25
26 % test type
27 mode = 'sequence';    % 'hole1' - hole 1 (far right)
28                       % 'hole5' - hole 5 (middle)
29                       % 'hole9' - hole 9 (far left)
30                       % 'sequence' - the sequence (right to left)
31
32 % ati address
```

```
33 atiAddress = '192.168.7.2';
34
35
36 % ------------------------------------
37 %% CONNECT TO ROBOT
38 % ------------------------------------
39 group = HebiLookup.newGroupFromFamily('MASCOT');
40 group.setFeedbackFrequency(1000); % [Hz]
41 group.setCommandLifetime(0.05);  % [s]
42 cmd = CommandStruct();
43 fbk = group.getNextFeedback();
44
45 gains = group.getGains();
46 gains.effortDeadZone = [0 0 0];
47 group.send('gains',gains);
48
49
50 % ------------------------------------
51 %% CONTROL GAINS & MODELS
52 % ------------------------------------
53
54 % starting tip location
55 switch mode
56     case 'hole1'
57         xd = [0.3820 -0.1380 0.25825 0 0 0]';
58     case 'hole5'
59         xd = [0.3770 0.0120 0.25825 0 0 0]';
60     case 'hole9'
61         xd = [0.3750 0.1610 0.25825 0 0 0]';
```

```
62      case 'sequence'
63          xd = [0.1000 -0.1500 0.25825 0 0 0]';
64  end
65
66  % load the robot kinematics representation
67  [kinematicsTool, kinematicsHandle, kinematicsAti,...
68      jacobianTool, jacobianHandle, jacobianAti] = loadKinematics();
69
70  % load the robot dyanamics representation
71  [mass, coriolis, damping] = loadDynamics();
72
73  % robot tool Kp
74  Ktp = diag([0 0 0 0 0 10]); % kp during run
75
76  % haptic device gains
77  Kdp = diag([1000 1000 1000 0 0 0]);
78  Kdd = diag([10 10 10 0 0 0]);
79  Jd0 = diag([-1 -1 1 -1 -1 1]); % coordinate transform
80
81  % Haptic device transform
82  if strcmp(mode,'sequence')
83      alpha = 5.0; % scale factor
84      xd0 = xd; % offset
85  else
86      alpha = 1.0; % scale factor
87      xd0 = xd + [-0.20 0 0 0 0 0]'; % offset
88  end
89
90  % handle forces
```

```
91 fh = zeros(6,1);
92
93
94 % ---------------------------------------
95 %% OPEN A LOG FILE
96 % ---------------------------------------
97 fid = fopen('datalog','w+');
98 fprintf(fid,'time,');
99 fprintf(fid,'sig_testRunning[0],');
100 fprintf(fid,'sig_hapticState[0],');
101
102 % motor data
103 fprintf(fid,'sig_jointFbkPos[0],sig_jointFbkPos[1],sig_jointFbkPos[2],');
104 fprintf(fid,'sig_jointFbkVel[0],sig_jointFbkVel[1],sig_jointFbkVel[2],');
105 fprintf(fid,'sig_jointFbkTrq[0],sig_jointFbkTrq[1],sig_jointFbkTrq[2],');
106
107 % raw ATI
108 fprintf(fid,'sig_atiFbkFrc[0],sig_atiFbkFrc[1],sig_atiFbkFrc[2],');
109 fprintf(fid,'sig_atiFbkFrc[3],sig_atiFbkFrc[4],sig_atiFbkFrc[5],');
110
111 % handle feedback
112 fprintf(fid,'sig_handleFbkPos[0],sig_handleFbkPos[1],sig_handleFbkPos[2],');
113 fprintf(fid,'sig_handleFbkPos[3],sig_handleFbkPos[4],sig_handleFbkPos[5],');
114 fprintf(fid,'sig_handleFbkVel[0],sig_handleFbkVel[1],sig_handleFbkVel[2],');
115 fprintf(fid,'sig_handleFbkVel[3],sig_handleFbkVel[4],sig_handleFbkVel[5],');
116 fprintf(fid,'sig_handleFbkFrc[0],sig_handleFbkFrc[1],sig_handleFbkFrc[2],');
117 fprintf(fid,'sig_handleFbkFrc[3],sig_handleFbkFrc[4],sig_handleFbkFrc[5],');
118 fprintf(fid,'sig_humnRefFrc[0],sig_humnRefFrc[1],sig_humnRefFrc[2],');
119 fprintf(fid,'sig_humnRefFrc[3],sig_humnRefFrc[4],sig_humnRefFrc[5],');
```

```
120 fprintf(fid,'\n');
121 fprintf(fid,'\n');
122
123 % --------------------------------------
124 %% MAIN LOOP
125 % --------------------------------------
126
127 % beeps for indicating test
128 SoundCountdown = sin(1:.2:1000);
129 SoundStart = sin(1:.3:5000);
130 SoundStop = sin(1:.2:2000);
131
132 % test flags and states
133 done = false;
134 test_running = false;
135 t  = 0; % runtime
136 t0 = 0; % test start
137 tc = 0; % countdown
138 n = 0; % number of demonstrations
139 Kdp_prev = zeros(6);
140 Kdd_prev = zeros(6);
141 Kd_filt = 0.001;
142
143 % run test
144 while(¬done)
145     tic
146
147     % --------------------------------------
148     % get feedback
```

```
149     fbk = group.getNextFeedback();
150     q = fbk.position.';
151     qd = fbk.velocity.';
152     trq = fbk.effort.';
153     xt = kinematicsTool(q); % tool position
154     xh = kinematicsHandle(q); % handle position
155     Jt = jacobianTool(q); % tool jacobian
156     Jh = jacobianHandle(q); % handle jacobian
157     Ja = jacobianAti(q); % ati jacobian
158     vt = Jt * qd;
159     vh = Jh * qd;
160
161     % get force-torque feedback
162     fm = atiConnection(atiAddress);
163     ft = Jh' \ Ja' * fm;
164
165     % ---------------------------------------
166     % get the haptic positions and velocities & compute force
167     fd = - (Jd0 \ fh) / alpha;
168     [xdev, vdev, bdev] = hapticConnection(fd);
169     xdev = alpha * Jd0 * xdev + xd0;
170     vdev = alpha * Jd0 * vdev;
171     if (bdev(2) > 0)
172         Kdp_prev = (1-Kd_filt) * Kdp_prev + (Kd_filt) * Kdp;
173         Kdd_prev = (1-Kd_filt) * Kdd_prev + (Kd_filt) * Kdd;
174     else
175         Kdp_prev = zeros(6);
176         Kdd_prev = zeros(6);
177     end
```

```
178     fh = Kdp_prev * (xdev - xh) + Kdd_prev * (vdev - vh);
179
180     % compute effort
181     ufb = Ktp * (xd - xt);
182
183     % apply
184     Gamma = Jt' * ufb + Jh' * fh;
185     cmd.effort = Gamma';
186     group.send(cmd);
187
188     % ---------------------------------------
189     % write data
190     fprintf(fid,'%.8f,',t);
191     fprintf(fid,'%i,',test_running);
192     fprintf(fid,'%i,',prod(bdev));
193
194     % motor data
195     fprintf(fid,'%.8f,%.8f,%.8f,',q(1),q(2),q(3));
196     fprintf(fid,'%.8f,%.8f,%.8f,',qd(1),qd(2),qd(3));
197     fprintf(fid,'%.8f,%.8f,%.8f,',trq(1),trq(2),trq(3));
198
199     % raw ATI
200     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
201         fm(1),fm(2),fm(3),fm(4),fm(5),fm(6));
202
203     % handle feedback
204     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
205         xh(1),xh(2),xh(3),xh(4),xh(5),xh(6));
206     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
```

```
207         vh(1),vh(2),vh(3),vh(4),vh(5),vh(6));
208     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
209         ft(1),ft(2),ft(3),ft(4),ft(5),ft(6));
210     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
211         fh(1),fh(2),fh(3),fh(4),fh(5),fh(6));
212     fprintf(fid,'\n');
213     % ------------------------------------
214
215     % check test condition
216     if test_running
217         if ((xt(1) < xd(1)) && (abs(vt(1)) < 0.005) && ((t-t0) > tmin))
218             test_running = false;
219         end
220     else
221         if n < Nd
222             tc = tc + toc;
223             if tc ≥Nc
224                 % start
225                 soundsc(SoundStart, 44100);
226                 test_running = true;
227                 t0 = t + toc;
228                 tc = 0;
229                 n = n + 1;
230                 fprintf('Starting demonstration %i\n',n);
231             else
232                 % countdown
233                 if floor(tc) ≠floor(t0)
234                     soundsc(SoundCountdown, 44100);
235                 end
```

```
236                 t0 = tc;
237             end
238         else
239             done = true;
240         end
241     end
242
243     % ---------------------------------------
244     % integrate time
245     et = toc;
246     disp(et)
247     t = t + et;
248     % ---------------------------------------
249 end
250
251 % end
252 soundsc(SoundStop, 44100);
253 fclose(fid);
```

Listing C.4: matlab/test_taskHardware.m - script for testing the learned policy on hardware

```
1 % Copyright 2014-2019, Parker Owan
2 % PhD Dissertation, University of Washington
3 %
4 % Script for testing the learned policy on hardware
5
6 clear all
7
8 % add GMM-GMR to path
```

```
addpath GMM-GMR-v2.0
addpath hebi
warning off


% --------------------------------------
%% OPTIONS
% --------------------------------------

% use acceleration feed forward
useAccelFF = false;

% use effort feed forward
useEffortFF = true;

% use force feedback compensator
useEffortFB = false;

% Number of tests
Ntests = 25;

% Test buffer (s)
Tbuffer = 0.1;

% approach time (s) - faster results in more tracking errors
Tapproach = 1.0;

% Target hole
hole = 5; % [1-9], 1 (right), 5 (center), 9 (left)
```

```
38
39 % Mahal threshold to start condition (was 1e-3)
40 distThreshold = 1e-2;
41 % distThreshold = 3.5e-1;
42 % distThreshold = 2e0;
43
44 % Use distance wait
45 waitForDistanceThreshold = false;
46
47 % Set the ati IP address
48 atiAddress = '192.168.7.2';
49
50 % set high if the prediction input and force feedback is filtered to 10 Hz
51 useFilteredPrediction = true;
52
53 % range of errors to be introduced (set to zero for no errors)
54 % This is the uniform domain (around center)
55 Ux = 0.03;          % (m)
56 Uy = 0;
57 % Uy = 0.02;
58 % Ux = 0.03;
59 % Uy = 0.015;
60 Ua = deg2rad(40);    % (rad) uniform
61
62 % nominal threshold - if we are this certain that an event is off-nominal,
63 % stop the task
64 NominalConfidence = 0.997;
65
66 % expected forces during shared autonomy
```

```
67 beta = 1;
68 Rf = beta * diag([30 20]);
69 humanTau = 0.1; % (s)
70
71 % hole location center (w.r.t. robot base)
72 d = 0.2252;
73 xc = [0.15284 + d, 0.0146];
74
75
76 % ---------------------------------------
77 %% CONNECT TO ROBOT
78 % ---------------------------------------
79 group = HebiLookup.newGroupFromFamily('MASCOT');
80 group.setFeedbackFrequency(1000); % [Hz]
81 group.setCommandLifetime(0.05);   % [s]
82 cmd = CommandStruct();
83 fbk = group.getNextFeedback();
84
85
86 % ---------------------------------------
87 %% CONTROL GAINS & MODELS
88 % ---------------------------------------
89
90 % sample rate
91 % dt = 1/1000; % (s) - discrete sample rate
92 dt = 1/200; % (s) - discrete sample rate
93
94 % load the robot kinematics representation
95 [kinematicsTool, kinematicsHandle, kinematicsAti,...
```

```
96      jacobianTool, jacobianHandle, jacobianAti] = loadKinematics();
97
98  % load the robot dyanamics representation
99  [mass, coriolis, damping] = loadDynamics();
100
101 % mask for active (non-singular variables)
102 activeChannels = boolean([1 1 0 0 0 0]);
103
104 % robot tip states
105 xtd = zeros(6,1);
106 vtd = zeros(6,1);
107
108 % robot tip Kp, Kd
109 Ktp = diag([0 0 0 0 0 10]);
110 Ktd = diag([0 0 0 0 0 0]);
111
112 % robot handle states
113 xhd = zeros(6,1); % desired handle position
114 vhd = zeros(6,1); % desired handle velocity
115 fhd = zeros(6,1); % feedforward handle force
116 xhe = zeros(6,1); % handle position integrator error
117 fha = zeros(6,1); % applied handle force (output from control law)
118
119 % robot handle Kp, Kd
120 Khp = diag([1250 1250 0 0 0 0]);
121 Khd = diag([10 10 0 0 0 0]); % was 1
122 Khi = diag([3e3 3e3 0 0 0 1]);
123 Khi_prev = zeros(6);
124 K_filt = 1e-3;
```

```
125 imax = [5 5 5 0.2 0.2 0.2]'; % max integrator value
126
127
128 % --------------------------------------
129 %% INITIAL CONDITIONS
130 % --------------------------------------
131
132 % hebi feedback
133 q = fbk.position.';
134 qd = fbk.velocity.';
135 xt = kinematicsTool(q); % tool pos
136 xh = kinematicsHandle(q); % handle pos
137 vt = [0 0 0 0 0 0]';
138 vh = [0 0 0 0 0 0]';
139 fm = atiConnection(atiAddress);
140
141
142 % --------------------------------------
143 %% Compute the target offset
144 % --------------------------------------
145
146 % get the relative distance between targets
147 dist_mu = 38.1809; % mm
148
149 % Note that we have a nice understanding of the relationship between all
150 % the holes. This relationship is captured here. Targets is (x,v).
151 % Training occured on (0, 0);
152 Targets = zeros(9,2);
153 for i = 1:9
```

```
154     Targets(i,2) = (i-5) * dist_mu / 1e3; % (m) distance between holes
155 end
156
157 % now compute the target offset - i.e. from the BrushKnet center
158 x_center = Targets(hole,:);
159
160
161 % --------------------------------------
162 %% INIT THE KNET
163 % --------------------------------------
164
165 % load
166 load('knet_737329.5889.mat')
167
168 % update the offset with the knet center
169 x_center = x_center + BrushKnet.mCenter.Mu;
170
171 % setup the internal states
172 BrushKnet.setup();
173
174 % initialize
175 BrushKnet.policyInit(0, 0);
176
177 % get the starting position
178 [x_start, ~] = BrushKnet.policyStep(0);
179
180 % offset by the goal location
181 v_start = [0 0];
182
```

```
183 % target location offset (error will be added later)
184 x_offset = x_center';
185
186 % We need to filter the input and output data that we feed to the
187 % prediction to whatever we used for training
188
189 % s - filter time constant (this is what we trained the model on)
190 tau = 0.1 / (2 * pi);
191 a_model = dt / (dt + tau);
192 optn = bodeoptions;
193 optn.Grid = 'on';
194 optn.FreqUnits = 'Hz';
195 sys = ss(-1/tau,1/tau,1,0);
196 sysd = ss(1-a_model,a_model,1,0,dt);
197 % bode(sys, sysd, optn)
198
199 % ---------------------------------------
200 %% Generate a minimum jerk trajectory to the starting position
201 % ---------------------------------------
202
203 % get the target states
204 x0 = xh(1:2);
205 x1 = x_start' + x_center';
206 v0 = vh(1:2);
207 v1 = v_start';
208 a0 = zeros(2,1);
209 a1 = zeros(2,1);
210
211 % generate
```

```
212 tg_dt = 1/50;
213 [tg_t, tg_x, tg_v, tg_a] = trajGenerator(x0, x1,...
214     v0, v1, a0, a1, [], [], tg_dt, 2);
215 [tga_t, tga_x, tga_v, tga_a] = trajGenerator(xtd(6), xtd(6),...
216     0, 0, 0, 0, [], [], tg_dt, 2);
217
218 % --------------------------------------
219 %% Open a log file
220 % --------------------------------------
221
222 % open
223 fid = fopen('datalog','w+');
224
225 % test info
226 fprintf(fid,'time,');
227 fprintf(fid,'sig_testRunning[0],');
228 fprintf(fid,'sig_actionIndex[0],');
229
230 % motor data
231 fprintf(fid,'sig_jointFbkPos[0],sig_jointFbkPos[1],sig_jointFbkPos[2],');
232 fprintf(fid,'sig_jointFbkVel[0],sig_jointFbkVel[1],sig_jointFbkVel[2],');
233 fprintf(fid,'sig_jointFbkTrq[0],sig_jointFbkTrq[1],sig_jointFbkTrq[2],');
234
235 % raw ATI
236 fprintf(fid,'sig_atiFbkFrc[0],sig_atiFbkFrc[1],sig_atiFbkFrc[2],');
237 fprintf(fid,'sig_atiFbkFrc[3],sig_atiFbkFrc[4],sig_atiFbkFrc[5],');
238
239 % handle references and commands
240 fprintf(fid,'sig_handleRefPos[0],sig_handleRefPos[1],sig_handleRefPos[2],');
```

```
241 fprintf(fid,'sig_handleRefPos[3],sig_handleRefPos[4],sig_handleRefPos[5],');
242 fprintf(fid,'sig_handleRefVel[0],sig_handleRefVel[1],sig_handleRefVel[2],');
243 fprintf(fid,'sig_handleRefVel[3],sig_handleRefVel[4],sig_handleRefVel[5],');
244 % reference is uff
245 fprintf(fid,'sig_handleRefFrc[0],sig_handleRefFrc[1],sig_handleRefFrc[2],');
246 fprintf(fid,'sig_handleRefFrc[3],sig_handleRefFrc[4],sig_handleRefFrc[5],');
247 % command is uff + ufb
248 fprintf(fid,'sig_handleCmdFrc[0],sig_handleCmdFrc[1],sig_handleCmdFrc[2],');
249 fprintf(fid,'sig_handleCmdFrc[3],sig_handleCmdFrc[4],sig_handleCmdFrc[5],');
250 % expectation is E[f|x,v]
251 fprintf(fid,'sig_handleExpFrc[0],sig_handleExpFrc[1],sig_handleExpFrc[2],');
252 fprintf(fid,'sig_handleExpFrc[3],sig_handleExpFrc[4],sig_handleExpFrc[5],');
253 % filtered handle feedback force
254 fprintf(fid,'sig_handleFltFrc[0],sig_handleFltFrc[1],sig_handleFltFrc[2],');
255 fprintf(fid,'sig_handleFltFrc[3],sig_handleFltFrc[4],sig_handleFltFrc[5],');
256
257 % handle feedback
258 fprintf(fid,'sig_handleFbkPos[0],sig_handleFbkPos[1],sig_handleFbkPos[2],');
259 fprintf(fid,'sig_handleFbkPos[3],sig_handleFbkPos[4],sig_handleFbkPos[5],');
260 fprintf(fid,'sig_handleFbkVel[0],sig_handleFbkVel[1],sig_handleFbkVel[2],');
261 fprintf(fid,'sig_handleFbkVel[3],sig_handleFbkVel[4],sig_handleFbkVel[5],');
262 fprintf(fid,'sig_handleFbkFrc[0],sig_handleFbkFrc[1],sig_handleFbkFrc[2],');
263 fprintf(fid,'sig_handleFbkFrc[3],sig_handleFbkFrc[4],sig_handleFbkFrc[5],');
264 fprintf(fid,'\n');
265
266
267 % --------------------------------------
268 %% MAIN LOOP
269 % --------------------------------------
```

```
270
271 % set up results struct to record errors that resulted in nominal or
272 % off-nominal events during the policy
273 Results.nominal = [];
274 Results.offnominal = [];
275
276 % filter states
277 fc_filt = 10; % (Hz) cutoff frequency
278 tau_filt = 1 / (2 * pi * fc_filt); % (s) time constant
279 filt_frcFbk = zeros(2,1);
280 filt_frcMu = zeros(2,1);
281 filt_frcCov = zeros(2,2);
282 ft_filt = zeros(6,1);
283
284 % set the first bias low
285 x_error = [0 0]';
286
287 % orientation bias
288 % Rot = eye(2);
289 % theta = deg2rad(10); % rad
290
291 % initialize filter states
292 dt_mean = dt;
293
294 % test counter
295 test_counter = 1;
296
297 % control loop
298 t0 = 0; % time offset call to run tg
```

```
299 tc = 0; % test countdown time
300 t = 0;
301 w = 0;
302 done = false;
303 test_running = false;
304 test_complete = false;
305 runTg = true;
306 runKnet = false;
307 actionIndex = 0;
308 nominalCondition = 1;
309 useKI = false; % start with integrator off for trajectory
310 % group.startLog();
311 while(¬done)
312
313     % start the compute couter
314     tic
315
316     % ---------------------------------------
317     % Control feedback
318     % ---------------------------------------
319     fmeas = atiConnection(atiAddress);
320     fbk = group.getNextFeedback();
321     q = fbk.position.';
322     qd = fbk.velocity.';
323     trq = fbk.effort.';
324     fm = fmeas;
325
326     % compute kinematics and jacobians
327     xt = kinematicsTool(q); % tool pos
```

```
328     xh = kinematicsHandle(q); % handle pos
329     Jt = jacobianTool(q);
330     Jh = jacobianHandle(q);
331     Ja = jacobianAti(q);
332     vt = Jt * qd; % tool vel from jacobian def
333     vh = Jh * qd; % tool vel from jacobian def
334     Jrobot = mass(q);
335
336     % compute the sensed forces in the handle frame
337     ft = Jh' \ Ja' * fmeas;
338
339     % integrate work
340     w = w + dot(vh(activeChannels),-ft(activeChannels)) * dt_mean;
341
342
343     % ---------------------------------------
344     % Run the trajectory generator/gain schedule integrator
345     % ---------------------------------------
346     if (runTg)
347
348         if ((t-t0) ≤tg_t(end))
349
350             % if the TG is running, find the position
351             xhd(1:2) = interp1(tg_t', tg_x', t-t0)';
352             vhd(1:2) = interp1(tg_t', tg_v', t-t0)';
353             xtd(6)  = interp1(tga_t', tga_x', t-t0)';
354         else
355
356             % otherwise, wait until we're close to start
```

```
357             if (waitForDistanceThreshold)
358                 ai = BrushKnet.mInternalStates.ActionIndex; % action index
359                 ci = BrushKnet.mActions{ai}(1); % cluster index
360                 ch = BrushKnet.mChannels; % knet channels
361                 s_mu = BrushKnet.mGmm.mu(ci,[ch.x ch.v]);
362                 s_sig = BrushKnet.mGmm.Sigma([ch.x ch.v],[ch.x ch.v],ci);
363
364                 % add in target location normalization
365                 s_mu(ch.x) = s_mu(ch.x) + x_offset';
366                 s_mu(ch.v) = 0;
367
368                 % check the error
369                 ed = s_mu' - [xh(1:2); vh(1:2)];
370
371                 % check if we're close enough to start
372                 dist = (ed' * inv(s_sig) * ed);
373                 if dist < distThreshold
374
375                     % show
376                     fprintf('Starting test %i of %i\n',test_counter,Ntests);
377
378                     % start the knet
379                     runTg = false;
380                     runKnet = true;
381                     useKI = false;
382                     test_running = true;
383                     w = 0;
384                     BrushKnet.policyInit(t, w);
385                 else
```

```
386
387                 % use integral control to get us close enough
388                 useKI = true;
389             end
390
391         % otherwise just go for it
392         else
393
394             % show
395             fprintf('Starting test %i of %i\n',test_counter,Ntests);
396
397             % start the knet
398             runTg = false;
399             runKnet = true;
400             useKI = false;
401             test_running = true;
402             w = 0;
403             BrushKnet.policyInit(t, w);
404         end
405     end
406 end
407
408 % ---------------------------------------
409 % Kinetic net controller
410 % ---------------------------------------
411 fmu = zeros(6,1);
412 f_mu = [0 0];
413 f_cov = Rf;
414 if (runKnet)
```

```
415
416         % step the policy
417         [xp, vp, ap, fp] = BrushKnet.policyStep(t, w);
418
419         % only use the X-channel prediction
420         xp(2) = 0;
421         vp(2) = 0;
422         fp(2) = 0;
423         ap(2) = 0;
424
425         % send the commands
426         xhd(activeChannels) = xp + x_offset'; % target location normalization
427         vhd(activeChannels) = vp;
428         fhd(activeChannels) = 0;
429         if (useAccelFF)
430             Mh = inv( Jh(activeChannels,:) / Jrobot * Jh(activeChannels,:)' );
431             fhd(activeChannels) = fhd(activeChannels) + Mh * ap';
432         end
433         if (useEffortFF)
434             fhd(activeChannels) = fhd(activeChannels) - fp';
435         end
436         actionIndex = BrushKnet.mInternalStates.ActionIndex;
437
438         % also, compute the expected value of the forces
439         xv = [xh(1:2)' - x_offset', vh(1:2)'];
440         fmu(1:2) = knet.predictForce(0, xv,...
441             BrushKnet.mActions{actionIndex},...
442             BrushKnet.mGmm, BrushKnet.mChannels);
443         f_mu = fmu(1:2);
```

```
444
445         % if the knet is finished, we're done with the test
446         if (BrushKnet.mInternalStates.Done)
447             test_complete = true;
448             Results.nominal = [Results.nominal, [x_error; xtd(6)]];
449         end
450     end
451
452     % ---------------------------------------
453     % Filter the feedback data (for prediction)
454     % ---------------------------------------
455     at = dt / humanTau;
456     Amatrix = [ 1-at, at; at, 1-at];
457     a_filt = dt / (dt + tau_filt);
458     filt_frcFbk = (1-a_filt) * filt_frcFbk + (a_filt) * ft(1:2);
459     filt_frcMu = (1-a_filt) * filt_frcMu + (a_filt) * f_mu;
460     filt_frcCov = (1-a_filt) * filt_frcCov + (a_filt) * f_cov;
461     if (test_running)
462         eta = Amatrix * [nominalCondition; 1-nominalCondition];
463         nominalLikelihood = knet.gaussian(filt_frcFbk(1),...
464             filt_frcMu(1), filt_frcCov(1,1));
465         lik = [nominalLikelihood; 1-nominalLikelihood];
466         eta = lik .* eta;
467         eta = eta ./ sum(eta);
468         nominalCondition = eta(1);
469
470         % full filtered force feedback
471         ft_filt = (1-a_model) * ft_filt + (a_model) * ft;
472
```

```
473         % if we're off-nominal, we're done with the test
474         if nominalCondition < (1-NominalConfidence)
475             test_complete = true;
476             Results.offnominal = [Results.offnominal, [x_error; xtd(6)]];
477         end
478     else
479         nominalCondition = 1;
480     end
481 
482     % -------------------------------------
483     % Integral controller
484     % -------------------------------------
485     if useKI
486         Khi_prev = (1-K_filt) * Khi_prev + (K_filt) * Khi;
487         xhe = xhe + dt_mean * (xhd - xh); % integrate
488         xhe = max(min(xhe,imax),-imax); % limit
489     else
490         Khi_prev = (1-K_filt) * Khi_prev + (K_filt) * zeros(6);
491         xhe = (1-K_filt) * xhe + (K_filt) * zeros(6,1); % filter to zero
492     end
493     fhi = Khi * xhe;
494 
495 
496     % -------------------------------------
497     % Compute the operational space controllers
498     % -------------------------------------
499 
500     % effort fb compensator
501     ffb = zeros(6,1);
```

```
502     if useEffortFB
503         ffb = (-ft) - fhd;
504     end
505
506     % rotate the handle position offset (about the hole location)
507 %   disp([xhd(1) xhd(2) xtd(6)])
508     Rot = [cos(xtd(6)) -sin(xtd(6)); sin(xtd(6)) cos(xtd(6))];
509     xhd(1:2) = Rot * (xhd(1:2) - xc') + xc';
510
511     % compute the handle controller
512     fha = (ffb + fhd + Khp * (xhd - xh) + Khd * (vhd - vh) + fhi);
513     uhandle = Jh' * fha;
514
515     % compute the tip controller
516     utip = Jt' * (Ktp * (xtd - xt) + Ktd * (vtd - vt));
517
518     % operational space control
519     Gamma = utip' + uhandle';
520     cmd.effort = Gamma;
521
522     % apply
523     group.send(cmd);
524
525
526     % --------------------------------------
527     % write data to test log
528     % --------------------------------------
529     % test info
530     fprintf(fid,'%.8f,',t);
```

```
531     fprintf(fid,'%i,',test_running);
532     fprintf(fid,'%i,',actionIndex);
533
534     % motor data
535     fprintf(fid,'%.8f,%.8f,%.8f,',q(1),q(2),q(3));
536     fprintf(fid,'%.8f,%.8f,%.8f,',qd(1),qd(2),qd(3));
537     fprintf(fid,'%.8f,%.8f,%.8f,',trq(1),trq(2),trq(3));
538
539     % raw ATI
540     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
541         fm(1),fm(2),fm(3),fm(4),fm(5),fm(6));
542
543     % handle references and commands
544     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
545         xhd(1),xhd(2),xhd(3),xhd(4),xhd(5),xhd(6));
546     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
547         vhd(1),vhd(2),vhd(3),vhd(4),vhd(5),vhd(6));
548     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
549         fhd(1),fhd(2),fhd(3),fhd(4),fhd(5),fhd(6));
550     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
551         fha(1),fha(2),fha(3),fha(4),fha(5),fha(6));
552     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
553         fmu(1),fmu(2),fmu(3),fmu(4),fmu(5),fmu(6));
554     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
555         ft_filt(1),ft_filt(2),ft_filt(3),ft_filt(4),ft_filt(5),ft_filt(6));
556
557     % handle feedback
558     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
559         xh(1),xh(2),xh(3),xh(4),xh(5),xh(6));
```

```
560     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
561         vh(1),vh(2),vh(3),vh(4),vh(5),vh(6));
562     fprintf(fid,'%.8f,%.8f,%.8f,%.8f,%.8f,%.8f,',...
563         ft(1),ft(2),ft(3),ft(4),ft(5),ft(6));
564
565     % newline
566     fprintf(fid,'\n');
567
568
569     % ---------------------------------------
570     % Test logic
571     % ---------------------------------------
572     if test_complete
573         tc = 0;
574         xtd(6) = 0; % needed to reset the damn TG
575         test_running = false; % set low
576         test_complete = false; % set low
577         test_counter = test_counter + 1; % update test counter
578         nominalCondition = 1; % reset nominal condition
579         runKnet = false;
580         actionIndex = 0;
581         BrushKnet.policyInit(0, 0); % reset the states
582         if test_counter > Ntests
583             done = true;
584         end
585     end
586
587     if (¬test_running && ¬runTg)
588         tc = tc + dt_mean;
```

```
589         % generate trajectory and run
590         if (tc > Tbuffer)
591             theta = Ua * (rand-0.5);
592             x_error = [Ux; Uy] .* (rand(2,1)-0.5);
593             x_offset = x_center' + x_error;
594             x1 = x_start' + x_offset;
595 
596             % generate tg for the handle
597             [tg_t0, tg_x0, tg_v0, tg_a0] = trajGenerator(xh(1:2), x0,...
598                 v0, v0, a0, a0, [], [], tg_dt, 2);
599             [tg_t1, tg_x1, tg_v1, tg_a1] = trajGenerator(x0, x1,...
600                 v0, v1, a0, a1, [], [], tg_dt, Tapproach);
601 
602             % generate tg for the orientation
603             [tga_t0, tga_x0, tga_v0, tga_a0] = trajGenerator(xtd(6), xtd(6),...
604                 0, 0, 0, 0, [], [], tg_dt, 2);
605             [tga_t1, tga_x1, tga_v1, tga_a1] = trajGenerator(xtd(6), theta,...
606                 0, 0, 0, 0, [], [], tg_dt, Tapproach);
607 
608             % cat the TG
609             tg_t = [tg_t0 tg_t1+tg_t0(end)+tg_dt];
610             tg_x = [tg_x0 tg_x1];
611             tg_v = [tg_v0 tg_v1];
612             tg_a = [tg_a0 tg_a1];
613             tga_t = [tga_t0 tga_t1+tga_t0(end)+tg_dt];
614             tga_x = [tga_x0 tga_x1];
615             tga_v = [tga_v0 tga_v1];
616             tga_a = [tga_a0 tga_a1];
617             runTg = true;
```

```
618             t0 = t;
619         end
620     end
621
622     % ---------------------------------------
623     % Control loop
624     % ---------------------------------------
625
626     % get elapsed
627     et = toc;
628
629     % estimate the mean sample rate
630     dt_mean = a_filt * et + (1-a_filt) * dt_mean;
631
632     % pause
633     pause(dt - et)
634
635     % update runtime
636     t = t + et + max(dt - et,0);
637
638 end
639 % log = group.stopLogFull();
640 fclose(fid);
641
642
643 %% plot the nominal / off-nominal results
644
645 figure(100),clf
646 hold on; grid on;
```

```
647 xlabel('Error $\hat{x}$ (m)')
648 ylabel('Error $\hat{y}$ (m)')
649 s1=scatter(Results.nominal(2,:),Results.nominal(3,:),75,...
650     'filled','k','MarkerFaceAlpha',1);
651 s2=scatter(Results.offnominal(2,:),Results.offnominal(3,:),75,...
652     'filled','r','MarkerFaceAlpha',1);
653 set(gcf,'position',[90 210 700 400]);
654 % xlim([-0.022 0.022])
655 % ylim([-0.010 0.010])
656 xl = xlim;
657 plot(xl,+3e-3*[1 1],'--k')
658 plot(xl,-3e-3*[1 1],'--k')
659 legend([s1 s2],'Nominal','Off-nominal','Location','NorthWest')
660
661
662 %% Plot the results of the test
663 data = parseLogFile('datalog');
664 figure(1),clf
665 h(1)=subplot(231);
666 hold on; grid on;
667 plot(data.time, data.handleRefPos(:,1),'-k','LineWidth',2)
668 plot(data.time, data.handleFbkPos(:,1),'-r','LineWidth',1)
669 ylabel('Position $\hat{x}$')
670 legend('Ref','Fbk')
671
672 h(2)=subplot(234);
673 hold on; grid on;
674 plot(data.time, data.handleRefPos(:,2),'-k','LineWidth',2)
675 plot(data.time, data.handleFbkPos(:,2),'-r','LineWidth',1)
```

```
676 ylabel('Position $\hat{y}$')
677 legend('Ref','Fbk')
678
679 h(3)=subplot(232);
680 hold on; grid on;
681 plot(data.time, data.handleRefVel(:,1),'-k','LineWidth',2)
682 plot(data.time, data.handleFbkVel(:,1),'-r','LineWidth',1)
683 ylabel('Velocity $\hat{x}$')
684 legend('Ref','Fbk')
685 ylim([-0.4 0.4])
686
687 h(4)=subplot(235);
688 hold on; grid on;
689 plot(data.time, data.handleRefVel(:,2),'-k','LineWidth',2)
690 plot(data.time, data.handleFbkVel(:,2),'-r','LineWidth',1)
691 ylabel('Velocity $\hat{y}$')
692 legend('Ref','Fbk')
693 ylim([-0.4 0.4])
694
695 h(5)=subplot(233);
696 hold on; grid on;
697 plot(data.time, data.handleRefFrc(:,1),'-k','LineWidth',2)
698 plot(data.time, -data.handleFbkFrc(:,1),'-r','LineWidth',1)
699 plot(data.time, -data.handleExpFrc(:,1),'-b','LineWidth',1)
700 plot(data.time, -data.handleFltFrc(:,1),'-g','LineWidth',1)
701 legend('Ref','Fbk','Pred','Flt fbk')
702 ylabel('Force $\hat{x}$')
703 ylim([-25 25])
704
```

```
705 h(6)=subplot(236);
706 hold on; grid on;
707 plot(data.time, data.handleRefFrc(:,2),'-k','LineWidth',2)
708 plot(data.time, -data.handleFbkFrc(:,2),'-r','LineWidth',1)
709 plot(data.time, -data.handleExpFrc(:,2),'-b','LineWidth',1)
710 plot(data.time, -data.handleFltFrc(:,2),'-g','LineWidth',1)
711 legend('Ref','Fbk','Pred','Flt fbk')
712 ylabel('Force $\hat{y}$')
713 ylim([-25 25])
714
715
716 % Plot the errors
717 figure(2),clf
718 h(7)=subplot(231);
719 hold on; grid on;
720 plot(data.time, data.handleRefPos(:,1)-data.handleFbkPos(:,1),...
721     '-r','LineWidth',2)
722 ylabel('Position $\hat{x}$')
723
724 h(8)=subplot(234);
725 hold on; grid on;
726 plot(data.time, data.handleRefPos(:,2)-data.handleFbkPos(:,2),...
727     '-r','LineWidth',2)
728 ylabel('Position $\hat{y}$')
729
730 h(9)=subplot(232);
731 hold on; grid on;
732 plot(data.time, data.handleRefVel(:,1)-data.handleFbkVel(:,1),...
733     '-r','LineWidth',2)
```

```
734 ylabel('Velocity $\hat{x}$')
735 ylim([-0.4 0.4])
736
737 h(10)=subplot(235);
738 hold on; grid on;
739 plot(data.time, data.handleRefVel(:,2)-data.handleFbkVel(:,2),...
740     '-r','LineWidth',2)
741 ylabel('Velocity $\hat{y}$')
742 ylim([-0.4 0.4])
743
744 h(11)=subplot(233);
745 hold on; grid on;
746 plot(data.time, data.handleRefFrc(:,1)-(-data.handleFbkFrc(:,1)),...
747     '-r','LineWidth',2)
748 plot(data.time, data.handleExpFrc(:,1)-(data.handleFltFrc(:,1)),...
749     '-b','LineWidth',1)
750 ylabel('Force $\hat{x}$')
751 ylim([-25 25])
752
753 h(12)=subplot(236);
754 hold on; grid on;
755 plot(data.time, data.handleRefFrc(:,2)-(-data.handleFbkFrc(:,2)),...
756     '-r','LineWidth',2)
757 plot(data.time, data.handleExpFrc(:,2)-(data.handleFltFrc(:,2)),...
758     '-b','LineWidth',1)
759 ylabel('Force $\hat{y}$')
760 ylim([-25 25])
761
762 linkaxes(h,'x')
```

```
763 % xlim([2 8])
764
765
766 % display some metrics
767 e_x = data.handleRefPos - data.handleFbkPos;
768 e_v = data.handleRefVel - data.handleFbkVel;
769 e_f = data.handleRefFrc - (-data.handleFbkFrc);
770 e_E1 = data.handleExpFrc - data.handleFltFrc;
771
772 mse_x = mean(sum(e_x(:,1:2).^2,2));
773 mse_v = mean(sum(e_v(:,1:2).^2,2));
774 mse_f = mean(sum(e_f(:,1:2).^2,2));
775 mse_E1 = mean(sum(e_E1(:,1:2).^2,2));
776
777 fprintf('==================================\n');
778 fprintf('Test results:\n');
779 fprintf('Hole: %i\n',hole);
780 fprintf('FF enabled: %i\n',useEffortFF);
781 fprintf('----------------------------------\n');
782 fprintf('X tracking MSE = %11.8f\n',mse_x);
783 fprintf('V tracking MSE = %11.8f\n',mse_v);
784 fprintf('F tracking MSE = %11.8f\n',mse_f);
785 fprintf('F prediction MSE = %11.8f\n',mse_E1);
786 fprintf('----------------------------------\n');
```

C.1.4 Subroutines and helper functions

Listing C.5: matlab/postProcessData.m - function for processing the data before learning

```
% Copyright 2014-2019, Parker Owan
% PhD Dissertation, University of Washington
function data = postProcessData(files, showFigs, fcutoff, fstop, dt,...
    dtTrain, poi, channels)
% data = postProcessData(files, showFigs, fcutoff, fstop, dt,...
%     dtTrain, poi, channels)
%
% Post process the test data set for use in knet
%
% files    - list of files to parse
% showFigs - boolean (true = show figures)
% fcutoff  - filter LPF cutoff frequency (Hz)
% fstop    - filter LPF stop frequency (Hz)
% dt       - interpolation sample rate (s)
% dtTrain  - sample rate for training data (s) - must be integer of dt
% poi      - position of interest ('handle', or 'tip')
% channels - select signal channels for compute
% data     - an array of data strctures


% design the filter
useFilter = true;
if (¬isempty(fcutoff) && ¬isempty(fstop))
    dfilt = designfilt('lowpassiir', ...
        'PassbandFrequency',2*fcutoff*dt,...
        'StopbandFrequency',2*fstop*dt, ...
        'PassbandRipple',1,'StopbandAttenuation',80);
else
    useFilter = false;
```

```
30      fprintf('Filter parameters not set. Not filtering the data!\n')
31  end
32
33  % load the kinematics objects
34  [kinematicsTool,kinematicsHandle,kinematicsAti,...
35      jacobianTool,jacobianHandle,jacobianAti] = loadKinematics();
36
37  % load the dynamics objects
38  [mass, coriolis, damping] = loadDynamics();
39
40  % Process the test files
41  Nsamples = 0;
42  for k = 1:length(files)
43
44      % ------------------------------------
45      % load the test data and normalize the time
46      d = load( files{k} );
47      t = d.time - d.time(1);
48
49      % We will only use the following raw data:
50      %       - joint position feedback (q)
51      %       - joint velocity feedback (qd)
52      %       - haptic force feedback (f)
53      % This is to make sure we're comparing the C++ code bit-for-bit
54      % with the matlab math, instead of taking the C++ at face value.
55      q = d.jointFbkPos;
56      qd = d.jointFbkVel;
57      fa = d.atiFbkFrc;
58      tr = d.jointFbkTrq;
```

```
59
60      % check for nans
61      idx1 = isnan(sum(q,2));
62      idx2 = isnan(sum(qd,2));
63      idx3 = isnan(sum(tr,2));
64      idt = idx1 | idx2 | idx3;
65
66      % remove nans
67      q(idt,:) = [];
68      qd(idt,:) = [];
69      fa(idt,:) = [];
70      tr(idt,:) = [];
71      t(idt,:) = [];
72
73      % interpolate
74      tn = (0:dt:t(end))';
75      q  = interp1(t,q,tn);
76      qd = interp1(t,qd,tn);
77      fa = interp1(t,fa,tn);
78      tr = interp1(t,tr,tn);
79      t = tn;
80
81      % -------------------------------------
82      % kinematics computations
83      xt = zeros(length(t),6);
84      xh = zeros(length(t),6);
85      vt = zeros(length(t),6);
86      vh = zeros(length(t),6);
87      fm = zeros(length(t),6);
```

```
88      fr = zeros(length(t),6);
89      for i = 1:length(t)
90          xt(i,:) = kinematicsTool(q(i,:))';
91          xh(i,:) = kinematicsHandle(q(i,:))';
92          vt(i,:) = qd(i,:) * jacobianTool(q(i,:))';
93          vh(i,:) = qd(i,:) * jacobianHandle(q(i,:))';
94          fm(i,:) = ((jacobianHandle(q(i,:))' \...
95              jacobianAti(q(i,:))') * fa(i,:)')';
96          fr(i,:) = jacobianHandle(q(i,:))' \ tr(i,:)';
97      end
98
99      % devlop the numerical derivative matrix
100     n = length(xt)-2;
101     fdiff = [-1, 1, zeros(1,n)];
102     cdiff = [diag(-1/2*ones(n,1)), zeros(n,2)] +...
103          [zeros(n,2), diag(1/2*ones(n,1))];
104     bdiff = [zeros(1,n), -1, 1];
105     Derivative = [fdiff; cdiff; bdiff] / dt;
106
107     % compute the accelerations
108     at = zeros(size(vt));
109     ah = zeros(size(vh));
110     for i = 1:6
111         at(:,i) = Derivative * vt(:,i); % tip acceleration
112         ah(:,i) = Derivative * vh(:,i); % handle acceleration
113     end
114
115     % ------------------------------------
116     % now filter the signals
```

```
117     if useFilter
118         Q  = filtfilt(dfilt, q); % joint angle
119         Qd = filtfilt(dfilt, qd); % joint velocity
120         Xt = filtfilt(dfilt, xt); % tip position
121         Xh = filtfilt(dfilt, xh); % handle position
122         Vt = filtfilt(dfilt, vt); % tip velocity
123         Vh = filtfilt(dfilt, vh); % handle velocity
124         At = filtfilt(dfilt, at); % tip accel
125         Ah = filtfilt(dfilt, ah); % handle accel
126         Fm = filtfilt(dfilt, fm); % measured force (at handle)
127         Fr = filtfilt(dfilt, fr); % robot force (at handle)
128     else
129         Q  = q; % joint angle
130         Qd = qd; % joint velocity
131         Xt = xt; % tip position
132         Xh = xh; % handle position
133         Vt = vt; % tip velocity
134         Vh = vh; % handle velocity
135         At = at; % tip accel
136         Ah = ah; % handle accel
137         Fm = fm; % measured force (at handle)
138         Fr = fr; % robot force (at handle)
139     end
140
141     % ---------------------------------------
142     % compute the mass
143     Lambda = zeros(length(t),6^2);
144     Faccel = zeros(size(Ah));
145     Atask = zeros(size(Ah));
```

```
146     Amotion = zeros(size(Ah));
147     for i = 1:length(t)
148         Jr = mass(q(i,:));
149         Mh = (jacobianHandle(q(i,:))' \ Jr) / jacobianHandle(q(i,:));
150         Lambda(i,:) = Mh(:)';
151         Faccel(i,:) = Mh * Ah(i,:)';
152         Atask(i,channels) = Mh(channels,channels) \ Fm(i,channels)';
153         Amotion(i,channels) = Ah(i,channels)' - Atask(i,channels)';
154     end
155
156     % ---------------------------------------
157     % push the data to the data struct
158     dat = knet.emptyDataStructure();
159
160     % time
161     dat.T = t;
162
163     % define training data
164     id_train = (1:dtTrain/dt:length(t))';
165     dat.id_cluster = id_train;
166
167     % define
168     switch poi
169         case 'handle'
170
171             % raw measurement data
172             dat.meas.X = xh(:,channels);
173             dat.meas.V = vh(:,channels);
174             dat.meas.A = ah(:,channels);
```

```
175             dat.meas.F = fm(:,channels);
176
177             % filtered handle kinematics
178             dat.X = Xh(:,channels);
179             dat.V = Vh(:,channels);
180             dat.A = Ah(:,channels);
181             dat.F = Fm(:,channels);
182
183         case 'tip'
184
185             % raw measurement data
186             dat.meas.X = xt(:,channels);
187             dat.meas.V = vt(:,channels);
188             dat.meas.A = at(:,channels);
189             dat.meas.F = fm(:,channels);
190
191             % filtered handle kinematics
192             dat.X = Xt(:,channels);
193             dat.V = Vt(:,channels);
194             dat.A = At(:,channels);
195             dat.F = Fm(:,channels);
196     end
197
198     % metadata
199     dat.name = files{k};
200     data(k) = dat;
201
202     % compute the number of samples
203     Nsamples = Nsamples + length(dat.id_cluster);
```

```
204 end
205
206 % ------------------------------------
207 %% PLOT THE PPT FIGS
208 % ------------------------------------
209
210 if (showFigs)
211     % plot the time history
212     figure(1),clf
213     hold on; grid on;
214     for k=1:length(data)
215         plot(data(k).T, data(k).X(:,1),'Color',[.3 .3 .3])
216     end
217     xlabel('time')
218     ylabel('position')
219     set(gca,'XTick',[])
220     set(gca,'YTick',[])
221     pos = get(gcf,'position');
222     set(gcf,'position',[pos(1:2) 560 180])
223     xlim([0 5])
224     ylim([0.1410   0.2214])
225
226     % plot the position-velocity
227     figure(2),clf
228     hold on; grid on;
229     for k=1:length(data)
230         plot(data(k).X(:,1), data(k).V(:,1),'Color',[.3 .3 .3])
231     end
232     xlabel('position')
```

```
233     ylabel('velocity')
234     set(gca,'XTick',[])
235     set(gca,'YTick',[])
236     pos = get(gcf,'position');
237     set(gcf,'position',[pos(1:2) 560 180])
238     xlim([0.1400   0.2300])
239     ylim([-0.4000   0.4000])
240
241     % plot the velocity-force
242     figure(3),clf
243     hold on; grid on;
244     for k=1:length(data)
245         plot(data(k).V(:,1), data(k).F(:,1),'Color',[.3 .3 .3])
246     end
247     xlabel('velocity')
248     ylabel('measured force')
249     set(gca,'XTick',[])
250     set(gca,'YTick',[])
251     pos = get(gcf,'position');
252     set(gcf,'position',[pos(1:2) 560 180])
253     xlim([-0.4000   0.4000])
254     ylim([-25.000  25.0000])
255
256     % plot the position-velocity-force
257     figure(4),clf
258     hold on; grid on;
259     for k=1:length(data)
260         plot3(data(k).X(:,1), data(k).V(:,1), data(k).F(:,1),...
261             'Color',[.3 .3 .3])
```

```
262     end
263     xlabel('position')
264     ylabel('velocity')
265     zlabel('measured force')
266     set(gca,'XTick',[])
267     set(gca,'YTick',[])
268     set(gca,'ZTick',[])
269     pos = get(gcf,'position');
270     set(gcf,'position',[pos(1:2) 560 180])
271     xlim([0.1400   0.2300])
272     ylim([-0.4000   0.4000])
273     zlim([-25.000  25.0000])
274     view(3)
275
276     % plot the position-velocity-force-acceleration
277     channel = 1;
278     figure(5),clf
279     subplot(221)
280     hold on; grid on;
281     for k=1:length(data)
282         plot(data(k).X(:,channel), data(k).A(:,channel), 'Color',[.3 .3 .3])
283     end
284     xlabel('position')
285     ylabel('acceleration')
286     set(gca,'XTick',[])
287     set(gca,'YTick',[])
288
289     subplot(222)
290     hold on; grid on;
```

```
291     for k=1:length(data)
292         plot(data(k).V(:,channel), data(k).A(:,channel), 'Color',[.3 .3 .3])
293     end
294     xlabel('velocity')
295     ylabel('acceleration')
296     set(gca,'XTick',[])
297     set(gca,'YTick',[])
298 
299     subplot(223)
300     hold on; grid on;
301     for k=1:length(data)
302         plot(data(k).X(:,channel), data(k).F(:,channel), 'Color',[.3 .3 .3])
303     end
304     xlabel('position')
305     ylabel('force')
306     set(gca,'XTick',[])
307     set(gca,'YTick',[])
308 
309     subplot(224)
310     hold on; grid on;
311     for k=1:length(data)
312         plot(data(k).V(:,channel), data(k).F(:,channel), 'Color',[.3 .3 .3])
313     end
314     xlabel('velocity')
315     ylabel('force')
316     set(gca,'XTick',[])
317     set(gca,'YTick',[])
318 
319 
```

```
320  % plot the position-velocity-force-acceleration
321  channel = 2;
322  figure(6),clf
323  subplot(221)
324  hold on; grid on;
325  for k=1:length(data)
326      plot(data(k).X(:,channel), data(k).A(:,channel), 'Color',[.3 .3 .3])
327  end
328  xlabel('position')
329  ylabel('acceleration')
330  set(gca,'XTick',[])
331  set(gca,'YTick',[])
332
333  subplot(222)
334  hold on; grid on;
335  for k=1:length(data)
336      plot(data(k).V(:,channel), data(k).A(:,channel), 'Color',[.3 .3 .3])
337  end
338  xlabel('velocity')
339  ylabel('acceleration')
340  set(gca,'XTick',[])
341  set(gca,'YTick',[])
342
343  subplot(223)
344  hold on; grid on;
345  for k=1:length(data)
346      plot(data(k).X(:,channel), data(k).F(:,channel), 'Color',[.3 .3 .3])
347  end
348  xlabel('position')
```

```
349     ylabel('force')
350     set(gca,'XTick',[])
351     set(gca,'YTick',[])
352 
353     subplot(224)
354     hold on; grid on;
355     for k=1:length(data)
356         plot(data(k).V(:,channel), data(k).F(:,channel), 'Color',[.3 .3 .3])
357     end
358     xlabel('velocity')
359     ylabel('force')
360     set(gca,'XTick',[])
361     set(gca,'YTick',[])
362 end
363 
364 return
```

Listing C.6: matlab/trajGenerator.m - function for generating minimum jerk trajectories

```
1  % Copyright 2014-2019, Parker Owan
2  % PhD Dissertation, University of Washington
3  function [time, pos, vel, acl, jrk] = trajGenerator(...
4      x0, xf, v0, vf, a0, af, vLim, aLim, dt, t)
5  % Generates a trajectory from starting vector position (x0), velocity (v0)
6  % and acceleration (a0) to final position (xf), velocity (vf)
7  % and acceleration (af) in time t. returns vectors of the trajectories at
8  % sample rate dt. vLim and aLim are not active.
9  
10     % If tf is not specified, assign unity time
```

```
11    if nargin < 6
12        t = 1.0;
13    end
14
15    % Compute the derivative matrix
16    A = [1 0 0 0 0 0;
17         0 1 0 0 0 0;
18         0 0 2 0 0 0;
19         1 t t^2 t^3 t^4 t^5;
20         0 1 2*t 3*t^2 4*t^3 5*t^4;
21         0 0 2 6*t 12*t^2 20*t^3];
22
23    % get the number of dimensions
24    n = length(x0);
25
26    % reshape the I/F.C. conditions (for row vector)
27    x0 = reshape(x0,[1 n]);
28    v0 = reshape(v0,[1 n]);
29    a0 = reshape(a0,[1 n]);
30    xf = reshape(xf,[1 n]);
31    vf = reshape(vf,[1 n]);
32    af = reshape(af,[1 n]);
33
34    % desired conditions
35    b = [x0; v0; a0; xf; vf; af];
36
37    % Get the coefficients
38    X = A \ b;
39
```

```
40      % Get the polynomials
41      time   = 0:dt:t;
42      T   = [time.^0; time.^1; time.^2; time.^3; time.^4; time.^5];
43
44      % Derivatives
45      vMult = repmat([1 2 3 4 5],[n 1]);
46      aMult = repmat([2 6 12 20],[n 1]);
47      jMult = repmat([6 24 60],[n 1]);
48
49      % Signals (taylor series)
50      pos = X' * T;
51      vel = (vMult .* X(2:end,:)') * T(1:end-1,:);
52      acl = (aMult .* X(3:end,:)') * T(1:end-2,:);
53      jrk = (jMult .* X(4:end,:)') * T(1:end-3,:);
54
55  return
```

Listing C.7: matlab/loadKinematics.m - function for loading robot kinematics

```
1  % Copyright 2014-2019, Parker Owan
2  % PhD Dissertation, University of Washington
3  function [Ft,Fh,Fm,Jt,Jh,Jm] = loadKinematics()
4  %
5  % F = @(q) function handle to tool point with q = configuration
6  % J = @(q) function handle to Jacobian with q = configuration
7
8  % parameters
9  h = 0.25825; % (m) height stack up
10 l1 = 0.16905; % (m) link 1
```

```
l2 = 0.18008; % (m) link 2
lt = 0.15520; % (m) tool
lh = -0.0700; % (m) handle
la = 0.04370; % (m) force torque sensor

% tool forward kinematics
Ft = @(q) [
    l1*cos(q(1)) + l2*cos(q(1)+q(2)) + lt*cos(q(1)+q(2)+q(3));
    l1*sin(q(1)) + l2*sin(q(1)+q(2)) + lt*sin(q(1)+q(2)+q(3));
    h;
    0;
    0;
    q(1)+q(2)+q(3);
    ];

% handle forward kinematics
Fh = @(q) [
    l1*cos(q(1)) + l2*cos(q(1)+q(2)) + lh*cos(q(1)+q(2)+q(3));
    l1*sin(q(1)) + l2*sin(q(1)+q(2)) + lh*sin(q(1)+q(2)+q(3));
    h;
    0;
    0;
    q(1)+q(2)+q(3);
    ];

% ati force/torque sensor forward kinematics
Fm = @(q) [
    l1*cos(q(1)) + l2*cos(q(1)+q(2)) + la*cos(q(1)+q(2)+q(3));
    l1*sin(q(1)) + l2*sin(q(1)+q(2)) + la*sin(q(1)+q(2)+q(3));
```

```
40      h;
41      0;
42      0;
43      q(1)+q(2)+q(3);
44      ];
45
46  % tool jacobian
47  Jt = @(q) [
48      -l1*sin(q(1)) - l2*sin(q(1)+q(2)) - lt*sin(q(1)+q(2)+q(3)),...
49      -l2*sin(q(1)+q(2)) - lt*sin(q(1)+q(2)+q(3)),...
50      -lt*sin(q(1)+q(2)+q(3));
51      l1*cos(q(1)) + l2*cos(q(1)+q(2)) + lt*cos(q(1)+q(2)+q(3)),...
52      l2*cos(q(1)+q(2)) + lt*cos(q(1)+q(2)+q(3)),...
53      lt*cos(q(1)+q(2)+q(3));
54      0,...
55      0,...
56      0;
57      0,...
58      0,...
59      0;
60      0,...
61      0,...
62      0;
63      1,...
64      1,...
65      1
66      ];
67
68  % handle jacobian
```

```
Jh = @(q) [
    -l1*sin(q(1)) - l2*sin(q(1)+q(2)) - lh*sin(q(1)+q(2)+q(3)),...
    -l2*sin(q(1)+q(2)) - lh*sin(q(1)+q(2)+q(3)),...
    -lh*sin(q(1)+q(2)+q(3));
    l1*cos(q(1)) + l2*cos(q(1)+q(2)) + lh*cos(q(1)+q(2)+q(3)),...
    l2*cos(q(1)+q(2)) + lh*cos(q(1)+q(2)+q(3)),...
    lh*cos(q(1)+q(2)+q(3));
    0,...
    0,...
    0;
    0,...
    0,...
    0;
    0,...
    0,...
    0;
    1,...
    1,...
    1
    ];

% ati force/torque jacobian
Jm = @(q) [
    0,...
    0,...
    0;
    -l1*cos(q(1)) - l2*cos(q(1)+q(2)) - la*cos(q(1)+q(2)+q(3)),...
    -l2*cos(q(1)+q(2)) - la*cos(q(1)+q(2)+q(3)),...
    -la*cos(q(1)+q(2)+q(3));
```

```
98      -l1*sin(q(1)) - l2*sin(q(1)+q(2)) - la*sin(q(1)+q(2)+q(3)),...
99      -l2*sin(q(1)+q(2)) - la*sin(q(1)+q(2)+q(3)),...
100     -la*sin(q(1)+q(2)+q(3));
101     1,...
102     1,...
103     1;
104     0,...
105     0,...
106     0;
107     0,...
108     0,...
109     0
110     ];
```

Listing C.8: matlab/loadDynamics.m - function for loading robot dynamics

```
1  % Copyright 2014-2019, Parker Owan
2  % PhD Dissertation, University of Washington
3  function [M,C,B] = loadDynamics()
4  % From vector of joint angles q, and velocities qd,
5  % compute the dynamics of the robot
6  %
7  % M = @(q)   3 x 3 matrix
8  % C = @(q,qd) 3 x 1 coriolis vector
9  % B = @(qd)  3 x 1 damping vector
10 %
11 % M(q) \ddot{q} + C(q,qdot) + B(qdot) = \Gamma
12
13 % parameters
```

```
14 m1 = 0.68;     % (kg) mass link 1
15 m2 = 0.68;     % (kg) mass link 2
16 m3 = 0.375;    % (kg) mass link 3
17 I1 = 1870e-6; % (kg-m^2) inertia link 1
18 I2 = 1870e-6; % (kg-m^2) inertia link 2
19 I3 = 1200e-6; % (kg-m^2) inertia link 3
20 r1 = 0.136;    % (m) com link 1
21 r2 = 0.136;    % (m) com link 2
22 r3 = 0.014;    % (m) com link 3
23 l1 = 0.16905; % (m) link 1
24 l2 = 0.18008; % (m) link 2
25 l3 = 0.15520; % (m) tool
26
27 % construct the motor model
28 motrN = [765.75, 765.75, 765.75]';
29 motrB = [3.031, 3.031, 3.031]'*1e-7;
30 motrJ = [4.993, 4.993, 4.993]'*1e-8;
31 Jm = diag(motrJ .* motrN.^2);
32 Bm = diag(motrB .* motrN.^2);
33
34 % collected terms (2 x 2)
35 % a = I1 + I2 + m1*r1^2 + m2*(l1^2 + r2^2);
36 % b = m2*l1*r2;
37 % d = I2 + m2*r2^2;
38
39 a = I1 + I2 + I3 + l1^2*(m2 + m3) + m1*r1^2 + m2*r2^2 + m3*(l2^2 + r3^2);
40 b = l1*(l2*m3 + m2*r2);
41 d = l2*m3*r3;
42 e = l1*m3*r3;
```

```
f = I2 + I3 + m2*r2^2 + m3*(l2^2 + r3^2);
g = I3 + m3*r3^2;

% sines and cosines
% s1 = sin(q(1));
% s2 = sin(q(2));
% s3 = sin(q(3));
% c1 = cos(q(1));
% c2 = cos(q(2));
% c3 = cos(q(3));

% compute matrices (2 x 2)
% M = [a + 2*b*c2, d + b*c2;
%      d+b*c2, d];
% C = [-b*s2*qd(1)*qd(2), -b*s2*qd(2)*(qd(1) + qd(2));
%       b*s2*qd(1)^2, 0];

% M = @(q) [a + 2*b*cos(q(2)), d + b*cos(q(2));
%           d+b*cos(q(2)), d];
% C = @(q,qd) [-b*sin(q(2))*qd(1)*qd(2), -b*sin(q(2))*qd(2)*(qd(1) + qd(2));
%              b*sin(q(2))*qd(1)^2, 0];

% mass matrix (3 x 3)
M = @(q) [Jm(1) + a + 2*(b*cos(q(2)) + d*cos(q(3)) + e*cos(q(2)+q(3))),...
          f + b*cos(q(2)) + 2*d*cos(q(3)) + e*cos(q(2)+q(3)),...
          g + d*cos(q(3)) + e*cos(q(2)+q(3));
          f + b*cos(q(2)) + 2*d*cos(q(3)) + e*cos(q(2)+q(3)),...
          Jm(2) + g + f + 2*d*cos(q(3)),...
          g + d*cos(q(3));
```

```
72          g + d*cos(q(3)) + e*cos(q(2)+q(3)),...
73          g + d*cos(q(3)),...
74          Jm(3) + g
75          ];
76
77 % coriolis forces (3 x 1)
78 C = @(q,qd) [-(b*sin(q(2)) + e*sin(q(2)+q(3)))*qd(2)*(2*qd(1)+qd(2)) ...
79                - 2*(d*sin(q(3)) + e*sin(q(2)+q(3)))*(qd(1)+qd(2))*qd(3) ...
80                - (d*sin(q(3)) + e*sin(q(2)+q(3)))*qd(3)^2;
81             (b*sin(q(2)) + e*sin(q(2)+q(3)))*qd(1)^2 ...
82                - 2*d*sin(q(3))*(qd(1)+qd(2))*qd(3) ...
83                - d*sin(q(3))*qd(3)^2;
84             e*cos(q(3))*sin(q(2))*qd(1)^2 ...
85                + e*cos(q(2))*sin(q(3))*qd(1)^2 ...
86                + d*sin(q(3))*(qd(1) + qd(2))^2;
87            ];
88
89 % Damping vector (3 x 1)
90 B = @(qd) [Bm(1)*qd(1);
91           Bm(2)*qd(2);
92           Bm(3)*qd(3)
93           ];
```

Listing C.9: matlab/resample.m - function for systematic resampling the particle filter

```
1 % Copyright 2014-2019, Parker Owan
2 % PhD Dissertation, University of Washington
3 function [x,w] = resample(x,w)
4
```

```
5     x_particle = x;
6     w_particle = w;
7
8     % get number of particles
9     % Get Length of Particles
10    n=length(x_particle);
11
12    % Cumulative Sum of Particles
13    % w_particle=w_particle;
14    c=cumsum(w_particle);
15
16    % Compute u Vector
17    u=zeros(n,1);
18    u(1)=inv(n)*rand(1);
19    u(2:n)=u(1)+inv(n)*(1:n-1);
20
21    % Pre-allocate Index
22    index=zeros(n,1);
23
24    % Compute Index for Resampling
25    i=1;
26    for j=1:n
27        while u(j)>c(i)
28            i=i+1;
29        end
30        index(j)=i;
31    end
32
33    % Resampled Data
```

```
34     x=x_particle(index,:);
35     w=inv(n)*ones(n,1);
36
37 end
```

C.1.5 Mex functions

Listing C.10: matlab/atiConnection.cpp - Mex function for reading data from the ATI Mini-45

```
1  /*
2  * Copyright 2014-2019, Parker Owan
3  * PhD Dissertation, University of Washington
4  *
5  * atiConnection.cpp - connect to an ATI force/torque sensor over UDP (RDT)
6  *
7  * setenv('MW_MINGW64_LOC','C:\MinGW') % Local Variable for MinGW path
8  *
9  * Mex compile command in matlab :
10 *     >> mex -I<path to boost> atiConnection.cpp
11 *     >> mex -I/usr/local/include atiConnection.cpp
12 *     >> mex -IC:/barc/boost_1_66_0 atiConnection.cpp
13 *
14 * Using WinGW, make sure to includet the flag to ws2_32.lib using
15 *     >> mex -IC:/boost_1_66_0 -lws2_32 atiConnection.cpp
16 *
17 * Usage : from MATLAB
18 *     >> forces = atiConnection('192.168.1.1');
19 *
```

```
20 * This is a C++ MEX-file for MATLAB.
21 */
22
23 // MATLAB mex files
24 #include "mex.hpp"
25 #include "mexAdapter.hpp"
26
27 // NOTE: We need to define this so that we only use error code headers
28 // There are compiler flags to NOT make use of the compiled boost ...
       dependencies,
29 // i.e. -DBOOST_DATE_TIME_NO_LIB -DBOOST_REGEX_NO_LIB -DBOOST_SYSTEM_NO_LIB
30 #define BOOST_ERROR_CODE_HEADER_ONLY
31 #define BOOST_DATE_TIME_NO_LIB
32 #define BOOST_REGEX_NO_LIB
33 #define BOOST_SYSTEM_NO_LIB
34
35 #include <boost/system/error_code.hpp>
36 #include "boost/asio.hpp"
37 #include <boost/array.hpp>
38 #include <boost/bind.hpp>
39 #include <iostream>
40 #include <string>
41
42 // Command
43 #define CMD_STREAM 2 // Command code 2 starts streaming
44 #define CMD_BIAS 42 // Command code 42 sets the software bias
45 #define NUM_SAMPLES 1 // Will send 1 sample before stopping
46
47 #define SEND_MSG_LENGTH 8
```

```
48 #define RECV_MSG_LENGTH 36
49
50 // using the boost udp protocol
51 using boost::asio::ip::udp;
52
53 /* Typedefs used so integer sizes are more explicit */
54 typedef unsigned int uint32;
55 typedef int int32;
56 typedef unsigned short uint16;
57 typedef short int16;
58
59
60 class MexFunction : public matlab::mex::Function {
61
62 private:
63     boost::asio::io_context m_ioContext;
64     udp::socket m_socket;
65     udp::resolver m_resolver;
66     udp::resolver::results_type m_endpoints;
67     std::string m_address = "192.168.1.1";
68
69 public:
70
71     // constructor
72     MexFunction()
73         : m_socket(m_ioContext, udp::endpoint(udp::v4(), 0)),
74           m_resolver(m_ioContext)
75     {
76
```

```
77      }
78
79      // destructor
80      ¬MexFunction() {
81          // Delete some stuff
82      }
83
84      // Input/output operator
85      void operator()(matlab::mex::ArgumentList outputs,
86          matlab::mex::ArgumentList inputs) {
87
88          // check the input/output arguments
89          checkArguments(outputs, inputs);
90
91          // set the address from the input string
92          matlab::data::CharArray inChar = inputs[0];
93          m_address = inChar.toAscii();
94
95          // define the endpoint object
96          m_endpoints = m_resolver.resolve(udp::v4(), m_address, "49152");
97
98          // initialize the output arrary
99          matlab::data::TypedArray<double> D = request();
100
101         // move the out to outputs
102         outputs[0] = std::move(D);
103     }
104
105
```

```
106     // Request
107     matlab::data::TypedArray<double> request() {
108
109         // open the array factory
110         matlab::data::ArrayFactory factory;
111
112         // create the array
113         matlab::data::TypedArray<double> D =
114             factory.createArray<double>({ 6,1 }, { 0,0,0,0,0,0 });
115
116         // create the message request
117         // ATI - see section 9.1 in Net F/T user manual.
118         char request[SEND_MSG_LENGTH];
119         *(uint16*)&request[0] = htons(0x1234);
120         *(uint16*)&request[2] = htons(CMD_STREAM);
121         *(uint32*)&request[4] = htonl(NUM_SAMPLES);
122
123         // send the message to the socket
124         m_socket.send_to(boost::asio::buffer(request, SEND_MSG_LENGTH),
125             *m_endpoints.begin());
126
127         // setup the receive buffer & endpoint
128         char reply[RECV_MSG_LENGTH];
129         udp::endpoint sender_endpoint;
130
131         // receive the message
132         size_t reply_length = m_socket.receive_from(
133             boost::asio::buffer(reply, RECV_MSG_LENGTH), sender_endpoint);
134
```

```
135         // parse the message
136         uint32 rdt_sequence = ntohl(*(uint32*)&reply[0]);
137         uint32 ft_sequence = ntohl(*(uint32*)&reply[4]);
138         uint32 status = ntohl(*(uint32*)&reply[8]);
139         int32 FTData[6];
140         for (int i = 0; i < 6; i++) {
141             FTData[i] = ntohl(*(int32*)&reply[12 + i * 4]);
142         }
143 
144         // print the message
145         // char * AXES[] = { "Fx", "Fy", "Fz", "Tx", "Ty", "Tz" };
146         // char * UNITS[] = { "N", "N", "N", "N-m", "N-m", "N-m" };
147         double C_FACTOR[6] = {
148             1.0 / 224808.9375,
149             1.0 / 224808.9375,
150             1.0 / 224808.9375,
151             1.0 / 8850746.0,
152             1.0 / 8850746.0,
153             1.0 / 8850746.0 };
154 
155         // Change the values.
156         int i = 0;
157         for (auto& elem : D) {
158             elem = C_FACTOR[i] * double(FTData[i]);
159             i++;
160         }
161         return D;
162     }
163 
```

```
164     // Check the input/output arguments
165     void checkArguments(matlab::mex::ArgumentList outputs,
166         matlab::mex::ArgumentList inputs) {
167         std::shared_ptr<matlab::engine::MATLABEngine> matlabPtr = getEngine();
168         matlab::data::ArrayFactory factory;
169 
170         if (inputs.size() != 1) {
171             std::string msg = "IP address required as input";
172             matlabPtr->feval(
173                 matlab::engine::convertUTF8StringToUTF16String("error"),
174                 0, std::vector<matlab::data::Array>({
175                     factory.createScalar(msg)
176             }));
177         }
178 
179         if (inputs[0].getType() != matlab::data::ArrayType::CHAR) {
180             std::string msg = "IP must be a string, e.g.: '192.168.1.1'";
181             matlabPtr->feval(
182                 matlab::engine::convertUTF8StringToUTF16String("error"),
183                 0, std::vector<matlab::data::Array>({
184                     factory.createScalar(msg)
185             }));
186         }
187     }
188 };
```

Listing C.11: matlab/hapticConnection.cpp - Mex function for interfacing to Force Dimension

```
/*
* Copyright 2014-2019, Parker Owan
* PhD Dissertation, University of Washington
*
* hapticConnection.cpp - connect to a Force Dimension haptic device
*
* Mex compile command in matlab :
*     >> mex -I<path to dhdc> -L<path to library> -ldhdms hapticConnection.cpp
*     >> mex '-I/Applications/dhd-3.2.2/include' \
*           '-L/Applications/dhd-3.2.2/build' -ldhd hapticConnection.cpp
*     >> mex '-IC:/Program Files/Force Dimension/sdk-3.6.0/include' \
*           '-LC:/Program Files/Force Dimension/sdk-3.6.0/lib' \
*           -ldhdms64 hapticConnection.cpp
*
* Usage : from MATLAB
*     >> [position, velocity, button] = hapticConnection(forces);
*
* This is a C++ MEX-file for MATLAB.
*/

// MATLAB mex files
#include "mex.hpp"
#include "mexAdapter.hpp"

// Force dimension sdk
#include "dhdc.h"

#include <string>

```

```
30 class MexFunction : public matlab::mex::Function {
31
32 private:
33     bool m_isConnected = false;
34
35
36 public:
37
38     // constructor
39     MexFunction()
40     {
41         // Connect to the force dimension
42         if (dhdOpen () ≥0) {
43             m_isConnected = true;
44             dhdEmulateButton(DHD_ON);
45         }
46     }
47
48     // destructor
49     ¬MexFunction() {
50         // Close the connection
51         dhdClose ();
52     }
53
54     // Input/output operator
55     void operator()(matlab::mex::ArgumentList outputs,
56         matlab::mex::ArgumentList inputs) {
57
58         // check the input/output arguments
```

```
59          checkArguments(outputs, inputs);
60
61          // set the address from the input string
62          matlab::data::TypedArray<double> forces = std::move(inputs[0]);
63
64          // apply the forces
65          applyForces(forces);
66
67          // get the feedback requests
68          matlab::data::TypedArray<double> X = getPositions();
69          matlab::data::TypedArray<double> V = getVelocities();
70          matlab::data::TypedArray<int> B = getButton();
71
72          // move the out to outputs
73          outputs[0] = std::move(X);
74          outputs[1] = std::move(V);
75          outputs[2] = std::move(B);
76      }
77
78      // Apply the device forces
79      void applyForces(matlab::data::TypedArray<double> D) {
80
81          // open the array factory
82          matlab::data::ArrayFactory factory;
83
84          // set the data.
85          double p[6] = { 0 };
86          int i = 0;
87          for (auto& elem : D) {
```

```
88              p[i] = elem;
89              i++;
90          }
91
92          if (m_isConnected) {
93              dhdSetForceAndTorque (p[0], p[1], p[2], p[3], p[4], p[5]);
94          }
95      }
96
97
98      // Request the device positions
99      matlab::data::TypedArray<double> getPositions() {
100
101         // open the array factory
102         matlab::data::ArrayFactory factory;
103
104         // create the array
105         matlab::data::TypedArray<double> D =
106             factory.createArray<double>({ 6,1 }, { 0,0,0,0,0,0 });
107
108         // get the positions
109         double p[6] = { 0 };
110         if (m_isConnected) {
111             dhdGetPosition (&p[0], &p[1], &p[2]);
112             dhdGetOrientationRad (&p[3], &p[4], &p[5]);
113         }
114
115         // Change the values.
116         int i = 0;
```

```
117         for (auto& elem : D) {
118             elem = p[i];
119             i++;
120         }
121 
122         // output the vector
123         return D;
124     }
125 
126 
127     // Request the device positions
128     matlab::data::TypedArray<int> getButton() {
129 
130         // open the array factory
131         matlab::data::ArrayFactory factory;
132 
133         // create the array
134         matlab::data::TypedArray<int> D =
135             factory.createArray<int>({ 2,1 }, { 0 });
136 
137         // get the data
138         int d[2] = { 0 };
139         d[0] = dhdGetButton(0);
140         int status[DHD_MAX_STATUS];
141         dhdGetStatus(status);
142         d[1] = status[DHD_STATUS_FORCE];
143 
144         // Change the data
145         int i = 0;
```

```
146         for (auto& elem : D) {
147             elem = d[i];
148             i++;
149         }
150 
151         // output the vector
152         return D;
153     }
154 
155 
156     // Request the device velocities
157     matlab::data::TypedArray<double> getVelocities() {
158 
159         // open the array factory
160         matlab::data::ArrayFactory factory;
161 
162         // create the array
163         matlab::data::TypedArray<double> D =
164             factory.createArray<double>({ 6,1 }, { 0,0,0,0,0,0 });
165 
166         // get the positions
167         double p[6] = { 0 };
168         if (m_isConnected) {
169             dhdGetLinearVelocity (&p[0], &p[1], &p[2]);
170             dhdGetAngularVelocityRad (&p[3], &p[4], &p[5]);
171         }
172 
173         // Change the values.
174         int i = 0;
```

```
175         for (auto& elem : D) {
176             elem = p[i];
177             i++;
178         }
179 
180         // output the vector
181         return D;
182     }
183 
184 
185     // Check the input/output arguments
186     void checkArguments(matlab::mex::ArgumentList outputs,
187         matlab::mex::ArgumentList inputs) {
188         std::shared_ptr<matlab::engine::MATLABEngine> matlabPtr = getEngine();
189         matlab::data::ArrayFactory factory;
190 
191         if (inputs.size() != 1) {
192             std::string msg = "A 6x1 force vector is required for input";
193             matlabPtr->feval(
194                 matlab::engine::convertUTF8StringToUTF16String("error"),
195                 0, std::vector<matlab::data::Array>({
196                     factory.createScalar(msg)
197             }));
198         }
199 
200         if (inputs[0].getType() != matlab::data::ArrayType::DOUBLE) {
201             std::string msg = "A 6x1 force vector is required for input";
202             matlabPtr->feval(
203                 matlab::engine::convertUTF8StringToUTF16String("error"),
```

```
204                 0, std::vector<matlab::data::Array>({
205                     factory.createScalar(msg)
206             }));
207         }
208 
209         if (inputs[0].getNumberOfElements() != 6) {
210             std::string msg = "A 6x1 force vector is required for input";
211             matlabPtr->feval(
212                 matlab::engine::convertUTF8StringToUTF16String("error"),
213                 0, std::vector<matlab::data::Array>({
214                     factory.createScalar(msg)
215             }));
216         }
217     }
218 };
```

C.1.6 MATLAB experiment runtime

Listing C.12: matlab/localizeTargetOnline.m - MATLAB Runtime particle filter localization

```
1 %  Copyright 2014-2019, Parker Owan
2 %  PhD Dissertation, University of Washington
3 
4 addpath GMM-GMR-v2.0
5 clear all
6 
7 % Font size (mac: 16, Windows: 12)
8 FontSize = 16;
9 
```

```
10 % Set the interpretter
11 warning off
12 set(0,'DefaultTextFontSize',FontSize,...
13       'DefaultAxesFontSize',FontSize,...
14       'DefaultTextInterpreter','Latex',...
15       'DefaultLegendInterpreter','Latex',...
16       'DefaultLineLineWidth',1,...
17       'DefaultLineMarkerSize',7.75)
18
19 % Import CoreRobotics
20 addpath('¬/Documents/CoreRobotics-1.1.0/matlab/lib')
21 import CoreRobotics.*
22
23 % ************************************************
24 %% OPTIONS
25 % ************************************************
26
27 % name of the learned file
28 model = 'knet_737329.5889.mat'; % using Xh - set poi = 'handle'
29
30 % position of interest ('handle' or 'tip')
31 poi = 'handle';
32
33 % Uniform region (square area over which the hole center could be)
34 % Ux = 0.21; % (m) ¬8 inches
35 Ux = 0.1; % (m) ¬4 inches
36 % Ux = 0.05; % (m) ¬2 inches
37 % Ux = 0.025; % (m) ¬1 inch
38
```

```
39 % Bias the initial estimate
40 useBias = true;
41
42 % data collection sample rate (s) (data gets interpolated to this)
43 % dt = 1/1000;
44 % dt = 1/200;
45 % dt = 1/100;
46 % dt = 1/20;
47 dt = 1/25;
48
49 % mask for active (non-singular variables)
50 activeChannels = boolean([1 1 0 0 0 0]);
51
52 % noise inflation factor (this is a tunable parameter)
53 betaX = 1e0; % (in x)
54 betaY = 1e0; % (in y)
55
56
57 % ************************************************
58 %% Load the memory map if to be used online
59 % ************************************************
60
61 % Open a shared memory object as client
62 memoryName = 'mascot-memory';
63 mascotMemory = CRSharedMemory(memoryName, CR_MANAGER_CLIENT);
64
65
66 % ************************************************
67 %% LOAD DATA AND MODELS
```

```
% ************************************************

% See: calibrate
Targets = [0.003063211714199 -0.152692929329856;
           0.002297408785650 -0.114519696997392;
           0.001531605857100 -0.076346464664928;
           0.000765802928550 -0.038173232332464;
                           0                  0;
          -0.000765802928550 0.038173232332464;
          -0.001531605857100 0.076346464664928;
          -0.002297408785650 0.114519696997392;
          -0.003063211714199 0.152692929329856];
load(model)

% get some members for convenience
ch = BrushKnet.mChannels;
gmm = BrushKnet.mGmm;

% define the contraint regions (i.e. error around the MAP estimate)
%
% This is two parts:
%   psi_1(y) : hole side constraint
%   psi_2(x) : coupon/brush length constraint
cthick = 0.00635; % (m) - coupon thickness (1/4 in)
rbar = 0.0032; % (m)
lstroke = -0.02; % (m)
rstroke = 0.03; %(m)
xbreve = max(gmm.mu(:,ch.x(1))) + (lstroke);
xbar = min(gmm.mu(:,ch.x(1))) + (xbreve - cthick);
```

```
97
98 % define the acceptible regin (true/false)
99 % x : [observations x 2]
100 % xHat : [estimates x 2]
101 acceptible_region = @(x, xHat) ((abs(x(:,2)-xHat(:,2)) ≤rbar) &...
102     (x(:,1)-xHat(:,1) ≥xbar) & (x(:,1)-xHat(:,1) ≤xbreve));
103
104 % define the hole center truth
105 center = BrushKnet.mCenter.Mu;
106
107
108 % ************************************************
109 %% INITIALIZE THE PARTICLE FILTER SETTINGS
110 % ************************************************
111
112 % noise inflation factor
113 beta = diag([betaX betaY]);
114 betaz = blkdiag(beta, beta, beta);
115
116 % get the covariances
117 Rx = beta * BrushKnet.mMeasurements.X;
118 Rv = beta * BrushKnet.mMeasurements.V;
119 Rf = beta * BrushKnet.mMeasurements.F;
120 Ra = beta * BrushKnet.mMeasurements.A;
121 Rz = blkdiag(Rx, Rv, Rf);
122
123 % bias the goal location error
124 if useBias
125     bias = [0.025, 0.004]; % (m)
```

```
126 else
127     bias = [0, 0];
128 end
129
130 % filter parameter (important to include this!)
131 fc = 2; % was 10 Hz (1 works great!)
132 sys = ss(-(2*pi*fc),(2*pi*fc),1,0);
133 sys1 = ss(-(2*pi*1),(2*pi*1),1,0);
134 sys2 = ss(-(2*pi*2),(2*pi*2),1,0);
135 sys3 = ss(-(2*pi*10),(2*pi*10),1,0);
136
137 sysd = c2d(sys,dt,'zoh');
138 sysd1 = c2d(sys1,dt,'zoh');
139 sysd2 = c2d(sys2,dt,'zoh');
140 sysd3 = c2d(sys3,dt,'zoh');
141 alpha = 1 - ssdata(sysd);
142 boptns = bodeoptions;
143 boptns.FreqUnits = 'Hz';
144 boptns.Grid = 'on';
145 % bode(sys,sysd,boptns)
146
147 % number of actions
148 na = length(BrushKnet.mActions);
149
150 % initialize a couple of grid elements (for marginalization)
151 nt = 1000;
152 np = 5000;
153 Xp = Ux * (rand(np,2) - 0.5) + center + bias;
154 Wp = ones(np,1) / np;
```

```
155
156 % random walk covariance
157 Qp = dt^2 * 5e-7 * eye(2); % from the AIM paper
158
159 % initial conditions
160 xh = mascotMemory.get('sig_handleFbkPos')';
161 vh = mascotMemory.get('sig_handleFbkVel')';
162 fh = mascotMemory.get('sig_handleFbkFrc')';
163 Xi = xh(1:2);
164 Vi = vh(1:2);
165 Fi = fh(1:2);
166
167
168 % ************************************************
169 %% BEGIN CONTROL LOOP
170 % ************************************************
171 done = false;
172 while ¬done
173
174     % start clock
175     tic
176
177     % ------------------------------------------------
178     % CHECK FOR RESET FLAG
179     % ------------------------------------------------
180     resetEstimator = mascotMemory.get('sig_resetEstimator');
181     if (resetEstimator)
182
183         % particle draws
```

```
184         Xp = Ux * (rand(np,2) - 0.5) + center + bias;
185         Wp = ones(np,1) / np;
186 
187         % initial conditions
188         xh = mascotMemory.get('sig_handleFbkPos')';
189         vh = mascotMemory.get('sig_handleFbkVel')';
190         fh = mascotMemory.get('sig_handleFbkFrc')';
191         Xi = xh(1:2);
192         Vi = vh(1:2);
193         Fi = fh(1:2);
194 
195         % estimate the position
196         estmTargetMu = Wp' * Xp;
197         estmTargetVar = var(Xp);
198 
199         % compute the confidence
200         cr = acceptible_region(Xp, estmTargetMu);
201         Confidence = Wp' * cr;
202 
203         % push updates to the memory map
204         tMu  = mascotMemory.get('sig_targetEstimate');
205         tVar = mascotMemory.get('sig_targetVariance');
206         tMu(1:2) = estmTargetMu;
207         tVar(1:2) = estmTargetVar;
208         mascotMemory.set('sig_targetEstimate', tMu);
209         mascotMemory.set('sig_targetVariance', tVar);
210         mascotMemory.set('sig_targetConfidence', Confidence);
211     end
212 
```

```
213
214     % ------------------------------------------------
215     % FEEDBACK
216     % ------------------------------------------------
217     % get the observation data
218     xh = mascotMemory.get('sig_handleFbkPos')';
219     vh = mascotMemory.get('sig_handleFbkVel')';
220     fh = mascotMemory.get('sig_handleFbkFrc')';
221
222     Ai = (vh(1:2) - Vi) / (dt / alpha);
223     Xi = (1 - alpha) * Xi + (alpha) * xh(1:2);
224     Vi = (1 - alpha) * Vi + (alpha) * vh(1:2);
225     Fi = (1 - alpha) * Fi + (alpha) * fh(1:2);
226     Zi = [Xi, Vi, Fi];
227     vars = [ch.x ch.v ch.f];
228
229     % get the hole index
230     holeIndex = mascotMemory.get('sig_holeIndex') + 1;
231
232     % ------------------------------------------------
233     % TARGET POSE ESTIMATION
234     % ------------------------------------------------
235     if (holeIndex > 0)
236
237         % random walk the target hypothesis
238         Xp = Xp + randn(np,2) * chol(Qp);
239
240         % compute the likelihood of each particle
241         xv = [Xi - (Targets(holeIndex,:) + Xp),...
```

```
242             ones(np,1) * Vi];
243
244         % for each action, predict the most likely forces
245         Lik = zeros(np,1);
246
247         % the brush stiffness model was found emprically
248         Kp = 3.5209e+03 * xv(:,1) + 131.5658;
249         Kp(xv(:,1)≤0) = 0;
250         fyhat = -Kp .* xv(:,2);
251         lf = mvnpdf(fyhat, Fi(2), 2);
252         lx = mvnpdf(0, xv(:,1), BrushKnet.mCenter.Sigma(1,1));
253
254         % now, for each action...
255         for a = 1:na
256
257             % get the manifold
258             man = BrushKnet.mActions{a};
259
260             % perhaps include the integrated motion (i.e. acceleration)
261             [amu, ¬] = BrushKnet.predictAccel(0, xv, man, gmm, ch);
262
263             % compute the likelihood
264             la = mvnpdf(amu, Ai, Ra);
265             lz = lf .* la .* lx;
266             Lik = max(Lik, lz);
267         end
268
269         % bayes update
270         Wp = Lik .* Wp;
```

```
271         Wp = Wp ./ sum(Wp);
272
273         % estimate the position
274         estmTargetMu = Wp' * Xp;
275         estmTargetVar = var(Xp);
276
277         % resample
278         Neff = 1./(Wp'*Wp);
279         if Neff < nt
280             [Xp, Wp] = resample(Xp, Wp);
281         end
282
283         % compute the confidence
284         cr = acceptible_region(Xp, estmTargetMu);
285         Confidence = Wp' * cr;
286
287         % push updates to the memory map
288         tMu  = mascotMemory.get('sig_targetEstimate');
289         tVar = mascotMemory.get('sig_targetVariance');
290         tMu(1:2) = estmTargetMu;
291         tVar(1:2) = estmTargetVar;
292         mascotMemory.set('sig_targetEstimate', tMu);
293         mascotMemory.set('sig_targetVariance', tVar);
294         mascotMemory.set('sig_targetConfidence', Confidence);
295
296         % display the target mean
297         disp(estmTargetMu)
298     end
299
```

```
300     % get elapsed
301     et = toc;
302
303     % pause
304     pause(dt - et)
305 end
```

C.2 Experiment code (C++)

C.2.1 Cmake Lists

Listing C.13: CMakeLists.txt

```
1  #  Copyright 2014-2019, Parker Owan
2  #  PhD Dissertation, University of Washington
3  #
4  #  CMake Lists for hole cleaning application
5  #
6  #  Dependencies:
7  #  Eigen, Boost, OpenCV, CoreRobotics, CHAI3d, and GTest (optional)
8  #  If these are in non-standard locations (i.e. not /usr/local),
9  #  then you need to set the paths so CMake can find these packages:
10 #  Windows: >> set CMAKE_PREFIX_PATH=...
11 #  Unix: >> set CMAKE_PREFIX_PATH=...
12 #
13 #  On Mac, this is easy:
14 #  >> cmake -G "Xcode" -DHEBI_DIR="¬/Documents/hebi" ..
15 #
16 #
```

```
17
18 # Output each path in the CMAKE_PREFIX_PATH
19 foreach(path ${CMAKE_PREFIX_PATH})
20     message("Path: " ${path})
21 endforeach(path)
22
23 cmake_minimum_required (VERSION 3.5)
24 project (app-brush)
25
26 # Set variables
27 SET (ROBOT_TARGET_NAME "app-brush")
28 SET (TEST_TARGET_NAME "app-tests")
29 SET (HEBINAME "hebi")
30
31 # Set compiler parameters
32 SET (CMAKE_CXX_STANDARD 11)
33 SET (CMAKE_CXX_STANDARD_REQUIRED ON)
34
35
36 # DEPENDENCIES
37
38 # Find external dependencies
39 find_package (OpenGL REQUIRED)
40 find_package (CoreRobotics 2.0.0 REQUIRED)
41 find_package (OpenCV REQUIRED)
42 find_package (CHAI3D REQUIRED)
43 find_package (tinyxml2 REQUIRED)
44 find_package (GTest)
45
```

```
# include the hebi cmake
if (NOT HEBI_DIR)
    message(FATAL_ERROR "HEBI_DIR not specified")
endif()
include(${HEBI_DIR}/cmake/hebi_platform.cmake)

# set the hebi library subdirectories
if(HEBI_HOST_LINUX)
  set(HEBI_C_LIB_SUBDIR "lib/linux_${HEBI_TARGET_ARCH}")
elseif(HEBI_HOST_WINDOWS)
  set(HEBI_C_LIB_SUBDIR "lib/win_${HEBI_TARGET_ARCH}")
elseif(HEBI_HOST_OSX)
  set(HEBI_C_LIB_SUBDIR "lib/osx_${HEBI_TARGET_ARCH}")
else()
  message(FATAL_ERROR "Unknown host platform")
endif()

# Windows global build options - for CHAI3d
message(STATUS ${CMAKE_SYSTEM_NAME})
if (CMAKE_SYSTEM_NAME MATCHES Windows)
    # VisualStudio compiler
    if (MSVC)
        add_definitions (-D_T_SECURE_NO_DEPRECATE)
        if (CMAKE_CL_64)
            add_definitions (-DWIN64)
        else ()
            add_definitions (-DWIN32)
        endif ()
    endif()
```

```
75 # Linux global build options
76 elseif (CMAKE_SYSTEM_NAME MATCHES Linux)
77     add_definitions (-DLINUX)
78 # Mac OS X global build options
79 elseif (CMAKE_SYSTEM_NAME MATCHES Darwin)
80     add_definitions (-DMACOSX)
81 endif ()
82
83 # Find GLUT/GL
84 if (CMAKE_SYSTEM_NAME MATCHES Windows)
85     find_package (FreeGLUT REQUIRED)
86 elseif (CMAKE_SYSTEM_NAME MATCHES Darwin)
87     find_package (GLUT REQUIRED)
88 endif ()
89
90 # header search paths
91 include_directories(
92     ${CMAKE_CURRENT_SOURCE_DIR}/src
93     ${CMAKE_CURRENT_SOURCE_DIR}/lib
94     ${CMAKE_CURRENT_SOURCE_DIR}/include
95     ${HEBI_DIR}/src
96     ${HEBI_DIR}/include
97     ${OpenCV_INCLUDE_DIRS}
98     ${CHAI3D_INCLUDE_DIRS}
99     ${OPENGL_INCLUDE_DIRS}
100    ${GLUT_INCLUDE_DIRS} #mac
101    ${FREEGLUT_INCLUDE_DIRS} #windows
102    ${CR_INCLUDE_DIRS}
103 )
```

```
104
105 # linker directories
106 link_directories(
107     ${CR_LIBRARY_DIRS}
108     ${CHAI3D_LIBRARY_DIRS}
109     ${HEBI_DIR}/${HEBI_C_LIB_SUBDIR}
110 )
111
112
113 # HEBI SETUP
114
115 # Add HEBI source files
116 # These are all of the source files required to
117 # build the C++ wrapper for the C API.
118 add_library(HEBI_CPP_FILES OBJECT
119     ${HEBI_DIR}/src/command.cpp
120     ${HEBI_DIR}/src/feedback.cpp
121     ${HEBI_DIR}/src/group_command.cpp
122     ${HEBI_DIR}/src/group.cpp
123     ${HEBI_DIR}/src/group_feedback.cpp
124     ${HEBI_DIR}/src/group_info.cpp
125     ${HEBI_DIR}/src/info.cpp
126     ${HEBI_DIR}/src/kinematics.cpp
127     ${HEBI_DIR}/src/lookup.cpp
128     ${HEBI_DIR}/src/mac_address.cpp
129     ${HEBI_DIR}/src/trajectory.cpp
130     ${HEBI_DIR}/src/log_file.cpp
131 )
132
```

```
133 # include the source code to build:
134 file(GLOB SOURCE_FILES
135     ${CMAKE_CURRENT_SOURCE_DIR}/src/*.cpp
136     ${CMAKE_CURRENT_SOURCE_DIR}/src/*.h
137     ${CMAKE_CURRENT_SOURCE_DIR}/src/*.hpp
138 )
139
140 # inlude the lib files to build
141 file(GLOB LIB_FILES
142     ${CMAKE_CURRENT_SOURCE_DIR}/lib/*.cpp
143     ${CMAKE_CURRENT_SOURCE_DIR}/lib/*.h
144     ${CMAKE_CURRENT_SOURCE_DIR}/lib/*.hpp
145     ${CMAKE_CURRENT_SOURCE_DIR}/include/*.h
146     ${CMAKE_CURRENT_SOURCE_DIR}/include/*.hpp
147 )
148
149 # HEBI turns off the exceptions, boost needs them enabled
150 IF(MSVC)
151     ADD_DEFINITIONS("/EHsc")
152 ENDIF(MSVC)
153
154
155 # BUILD
156
157 # Set the exectuable target
158 add_executable(${ROBOT_TARGET_NAME} ${SOURCE_FILES} ${LIB_FILES}
159     $<TARGET_OBJECTS:HEBI_CPP_FILES>)
160 target_link_libraries(${ROBOT_TARGET_NAME} ${OpenCV_LIBS}
161     ${CHAI3D_LIBRARIES} ${OPENGL_LIBRARIES} ${GLUT_LIBRARY}
```

```
162     ${FREEGLUT_LIBRARIES} ${CR_LIBRARIES} tinyxml2)
163
164 # hebi target links
165 if (WIN32)
166     target_link_libraries( ${ROBOT_TARGET_NAME} "hebi" kernel32 )
167 else()
168     target_link_libraries( ${ROBOT_TARGET_NAME} "hebi" m pthread )
169 endif()
170
171 # put the files into some groups
172 source_group("\\src" FILES ${SOURCE_FILES})
173 source_group("\\lib" FILES ${LIB_FILES})
174
175
176 # COPY DYNAMIC LIBRARIES
177
178 # Copy over any missing dlls we need to the target directory
179 # Copy the Hebi .dll
180 if (WIN32)
181     add_custom_command( TARGET ${ROBOT_TARGET_NAME} POST_BUILD
182         COMMAND ${CMAKE_COMMAND} -E copy_if_different
183         "${HEBI_DIR}/${HEBI_C_LIB_SUBDIR}/${HEBINAME}.dll"
184         $<TARGET_FILE_DIR:${ROBOT_TARGET_NAME}>)
185
186     # Glob the opencv .dlls
187     file(GLOB OPENCV_DLLS src/core/*.hpp )
188
189     # and generate a new copy for each
190     add_custom_command( TARGET ${ROBOT_TARGET_NAME} POST_BUILD
```

```
191         COMMAND ${CMAKE_COMMAND} -E copy_if_different
192         "${HEBI_DIR}/${HEBI_C_LIB_SUBDIR}/${HEBINAME}.dll"
193         $<TARGET_FILE_DIR:${ROBOT_TARGET_NAME}>)
194 endif()
195
196 # copy over dylib
197 if (APPLE)
198     add_custom_command( TARGET ${ROBOT_TARGET_NAME} POST_BUILD
199         COMMAND ${CMAKE_COMMAND} -E copy_if_different
200         "${HEBI_DIR}/${HEBI_C_LIB_SUBDIR}/libhebi.1.0.dylib"
201         "${HEBI_DIR}/${HEBI_C_LIB_SUBDIR}/libhebi.1.dylib"
202         "${HEBI_DIR}/${HEBI_C_LIB_SUBDIR}/libhebi.dylib"
203         $<TARGET_FILE_DIR:${ROBOT_TARGET_NAME}>)
204 endif()
205
206
207 # BUILD TESTS
208
209 # Get the test files
210 file(GLOB TESTS tests/*.cpp)
211
212 # Check for GTest
213 if(NOT GTEST_FOUND)
214     message(STATUS "GTEST not found, not building the test suite")
215
216 else()
217     message(STATUS "GTEST found, building the test suite")
218
219     # add the gtest includes
```

```
220     include_directories(${GTEST_INCLUDE_DIRS})
221
222     # Add an executable for the tests
223     add_executable(${TEST_TARGET_NAME} ${TESTS} ${LIB_FILES})
224
225     # Link the library binary
226     target_link_libraries(${TEST_TARGET_NAME}
227         ${GTEST_LIBRARIES} ${OpenCV_LIBS}
228         ${CHAI3D_LIBRARIES} ${OPENGL_LIBRARIES} ${GLUT_LIBRARY}
229         ${FREEGLUT_LIBRARIES} ${CR_LIBRARIES})
230
231 endif()
```

C.2.2 Runtime source

Listing C.14: src/main.h - Main header

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #ifndef main_h
7  #define main_h
8
9  // ------------------------------------------------
10 // Includes
11 // ------------------------------------------------
12
```

```
13 // include the control interfaces
14 #include "AtiFeedback.h"
15 #include "AutonomyStep.h"
16 #include "DynamicsStep.h"
17 #include "Trajectory.h"
18
19 // Include some standard stuff
20 #include <vector>
21 #include <string>
22 #include <fstream> // file stream
23 #include <sstream> // string stream
24 #include <iostream> // io
25 // #include <string>
26 // #include <cstdlib> //std::system
27
28 // C++ headers
29 #include <math.h>
30 #include <vector>
31
32 // Include math constants
33 #define _USE_MATH_DEFINES
34
35 // include Eigen
36 #include "Eigen/Dense"
37 #include "Eigen/Geometry"
38
39 // Include the external libraries
40 #include "CoreRobotics.hpp"
41 #include "chai3d.h"
```

```
42 #include "hebi.h"
43 #include "lookup.hpp"
44 #include "tinyxml2.h"
45
46 // OpenCV
47 #include <opencv2/core.hpp>
48 #include <opencv2/highgui.hpp>
49 #include <opencv2/videoio.hpp>
50
51 // OpenGL
52 #ifndef MACOSX
53 #include "GL/glut.h"
54 #else
55 #include "GLUT/glut.h"
56 #endif
57
58 // CONSTANTS
59 #define NUM_MODULES 3
60
61 // MEMORY ISSUES WITH BOOST...?
62 #define BOOST_INTERPROCESS_POSIX_PROCESS_SHARED
63
64 using namespace chai3d;
65
66 // ---------------------------------------------------
67 // Default settings
68 // ---------------------------------------------------
69
70 // Gains
```

```
71 double gKpd = 300.0;      // haptic device position gain (300)
72 double gKdd = 1.0;        // haptic device velocity gain (1)
73 double gGam = 3.5;        // force reflection scale factor (2.5)
74 double gKph = 1250.0;     // robot handle position gain
75 double gKdh = 10.0;       // robot handle velocity gain
76 double gKpt = 10.0;       // robot tip angle position gain
77
78 // IP ADDRESS FOR F/T SENSOR
79 std::string atiAddress = "192.168.7.2";
80
81 // viewer (sim) only
82 bool gSimulationOnly = false;
83
84 // test mode
85 OperatingMode gOperatingMode = OPERATING_MODE_AUTONOMOUS;
86
87 // Constants
88 double gTargetX = 0.185944997384898;
89 double gTargetY = 0.014569740682590;
90 double gTargetZ = 0.25825;
91 double gTargetVarX = 0.528588063977846e-3;
92 double gTargetVarY = 0.004948387413381e-3;
93 double gHoleOffsetFromTargetX = 0.1952;
94
95 // Camera index
96 unsigned gCameraIndex = 0;
97
98 // Camera size
99 unsigned gCameraWidth = 1280;    // 640
```

```
100 unsigned gCameraHeight = 720;    // 480
101
102 // Haptic workspace radius (m)
103 double gWorkspaceRadius = 0.4;   // 0.4
104
105 // Workspace center (x)
106 double gWorkspaceCenterX = 0.15; // 0.175
107
108 // Camera display transparency
109 double gCameraTransparency = 1.0;
110
111 // Instruction set (0: hole 5, 1: holes 1-9)
112 unsigned gInstructions = 1;      // 0, 1
113
114 // Hole locations (relative to the target center)
115 double gHoleFiducials[9][2] = {
116    0.003063211714199, -0.152692929329856,
117    0.002297408785650, -0.114519696997392,
118    0.001531605857100, -0.076346464664928,
119    0.000765802928550, -0.038173232332464,
120                     0,                  0,
121   -0.000765802928550, 0.038173232332464,
122   -0.001531605857100, 0.076346464664928,
123   -0.002297408785650, 0.114519696997392,
124   -0.003063211714199, 0.152692929329856};
125
126
127 // -------------------------------------------------
128 // Global custom objects
```

```
129 // ------------------------------------------------
130 AutonomyStep* gAutonomy;
131 cr::core::Clock gTimer;
132 cv::Mat gImageFrame;
133 cMutex gImageLock;
134
135 // ------------------------------------------------
136 // Global CoreRobotics objects
137 // ------------------------------------------------
138 // these are globals for now because CoreRobotics signals are incomplete
139 cr::physics::EulerMode gEulerConvention = cr::physics::CR_EULER_MODE_XYZ;
140 cr::world::NodePtr gTargetItem;
141 cr::world::NodePtr gWorldItem[10];
142 cr::world::RobotPtr gRobot;
143 cr::world::LinkPtr gRobotLink[4];
144 cr::world::NodePtr gRobotTool[4];
145 cr::control::TrajectoryGenerator gTrajGenerator;
146
147
148 // ------------------------------------------------
149 // global signal references
150 // ------------------------------------------------
151 Eigen::VectorXd gTargetLocation(6);
152 Eigen::VectorXd gTargetEstimate(6);
153 Eigen::VectorXd gTargetVariance(6);
154 Eigen::VectorXd gJointCmdTrq(3);
155 Eigen::VectorXd gJointFbkPos(3);
156 Eigen::VectorXd gJointFbkVel(3);
157 Eigen::VectorXd gJointFbkTrq(3);
```

```
158 Eigen::VectorXd gAtiFbkFrc(6);
159 Eigen::VectorXd gHandleRefPos(6);
160 Eigen::VectorXd gHandleRefVel(6);
161 Eigen::VectorXd gHandleRefFrc(6);
162 Eigen::VectorXd gHandleCmdFrc(6);
163 Eigen::VectorXd gHumnRefFrc(6);
164 Eigen::VectorXd gHandleFbkPos(6);
165 Eigen::VectorXd gHandleFbkVel(6);
166 Eigen::VectorXd gHandleFbkFrc(6);
167 double         gLevelOfAutonomy = 1; // 1 : autonomy, 0 : teleoperation
168 double         gLocalizationConf = 1;
169 bool           gTestRunning = false;
170 bool           gButtonPress = false;
171
172
173 // ---------------------------------------------------
174 // Global flags
175 // ---------------------------------------------------
176 bool gControlActive = false;
177 bool gFullscreen = false;
178 bool gSimulationRunning = false;
179 bool gRobotConnected = false;
180 bool gAtiConnected = false;
181 bool gCameraConnected = false;
182 bool gBilateralEnabled = false;
183
184
185 // ---------------------------------------------------
186 // CHAI3D
```

```
187 // ---------------------------------------------------
188
189 // these need to go to the haptic loop
190 cWorld* gHapticWorld;
191 cToolCursor* gHapticTool;
192 cGenericHapticDevicePtr gHapticDevice;
193
194 // these need to be global because they are passed to graphics
195 cCamera* camera;
196 cCamera* cameraTool; // virtual camera
197 cFrameBufferPtr frameBuffer; // buffer for virtual camera
198 cViewPanel* viewPanel; // side panel to display frame buffer
199 cScope* signalScope; // scope for data display
200 cLabel* labelHapticRate; // label for showing current rate
201 cLevel* levelVelocity; // level from [0,1] for state display
202 cPanel* messagePanel; // side panel to display message text
203 cLabel* labelMessage; // message text
204 cLabel* labelOperatingMode; // test mode text
205 cLabel* labelTestStatus; // test status text
206 cBitmap* imageBitmap; // camera frame display
207 cImagePtr imageDataPtr; // pointer to the image data
208 cMultiSegment* trajectorySegment; // trajectory line segment
209
210 // these are global for now because we still need to update signal structure
211 cMultiMesh* meshCadCoupon;
212 cMultiMesh* meshCadLink[4];
213 cShapeSphere* sphereTool[4];
214 cShapeSphere* sphereCenterOfMass[3];
215 cShapeSphere* sphereFiducial[9];
```

```
216 cShapeBox* boxFiducial[9];
217 cShapeSphere* sphereReference;
218
219 // frequency counter
220 chai3d::cFrequencyCounter gFrequencyCounter;
221
222 // information about computer screen and GLUT display window
223 int screenW;
224 int screenH;
225 int windowW;
226 int windowH;
227 int windowPosX;
228 int windowPosY;
229 int mouseX;
230 int mouseY;
231
232
233 // ---------------------------------------------------
234 // Helper functions
235 // ---------------------------------------------------
236
237
238 // callback when a key is pressed
239 void keySelect(unsigned char key, int x, int y)
240 {
241     // option ESC: exit
242     if ((key == 27) || (key == 'q'))
243     {
244         exit(0);
```

```
245     }
246
247     // option f: toggle gFullscreen
248     if (key == 'f')
249     {
250         if (gFullscreen)
251         {
252             windowPosX = glutGet(GLUT_INIT_WINDOW_X);
253             windowPosY = glutGet(GLUT_INIT_WINDOW_Y);
254             windowW = glutGet(GLUT_INIT_WINDOW_WIDTH);
255             windowH = glutGet(GLUT_INIT_WINDOW_HEIGHT);
256             glutPositionWindow(windowPosX, windowPosY);
257             glutReshapeWindow(windowW, windowH);
258             gFullscreen = false;
259         }
260         else
261         {
262             glutFullScreen();
263             gFullscreen = true;
264         }
265     }
266
267     // option k: toggle bilateral teleoperation enabled
268     if (key == 'k') {
269         if (gBilateralEnabled)
270         {
271             std::cout << "Bilateral disabled\n";
272             gBilateralEnabled = false;
273         }
```

```
        else
        {
            std::cout << "Bilateral enabled\n";
            gBilateralEnabled = true;
        }
    }

    // option space: toggle test running flag
    if (key == 't') {
        if (gOperatingMode == OPERATING_MODE_AUTONOMOUS)
        {
            gOperatingMode = OPERATING_MODE_TELEOPERATION;
        }
        else if (gOperatingMode == OPERATING_MODE_TELEOPERATION)
        {
            gOperatingMode = OPERATING_MODE_COLLABORATION;
        }
        else if (gOperatingMode == OPERATING_MODE_COLLABORATION)
        {
            gOperatingMode = OPERATING_MODE_AUTONOMOUS;
        }
    }

    // option space: toggle test running flag
    if (key == ' ') {
        if (gTestRunning)
        {
            gTestRunning = false;
        }
```

```
303         else
304         {
305             gTestRunning = true;
306         }
307     }
308 
309     // option b: move the sequence back
310     if (key == 'b') {
311         // std::cout << "Moved back a step\n";
312         int index = gAutonomy->getSequenceIndex();
313         index-=1;
314         gAutonomy->setSequenceIndex(index);
315     }
316 }
317 
318 // callback when the window display is updated
319 void updateGraphics(void) {
320 
321 
322     //----------------------------------------------------------
323     // UPDATE GRAPHICS OBJECTS
324     //----------------------------------------------------------
325 
326     // update the target sphere location
327     cr::physics::Frame f = gTargetItem->getLocalTransform();
328     Eigen::Vector3d p = f.getTranslation();
329     p(0) = gTargetEstimate(0);
330     p(1) = gTargetEstimate(1);
331     f.setTranslation(p);
```

```
332     gTargetItem->setLocalTransform(f);
333
334     // test coupon graphic update
335     meshCadCoupon->setLocalPos(
336         gWorldItem[0]->getGlobalTransform().getTranslation() );
337     meshCadCoupon->setLocalRot(
338         gWorldItem[0]->getGlobalTransform().getRotation() );
339
340     // fiducial graphic update
341     for (int i = 0; i < 9; i++){
342         sphereFiducial[i]->setLocalPos( ...
                gWorldItem[i+1]->getGlobalTransform().getTranslation() );
343         sphereFiducial[i]->setLocalRot( ...
                gWorldItem[i+1]->getGlobalTransform().getRotation() );
344         boxFiducial[i]->setLocalPos( ...
                gWorldItem[i+1]->getGlobalTransform().getTranslation() );
345         boxFiducial[i]->setLocalRot( ...
                gWorldItem[i+1]->getGlobalTransform().getRotation() );
346         boxFiducial[i]->setEnabled(gAutonomy->getHoleIndex() == i);
347     }
348
349     // link 1 cad graphic update
350     meshCadLink[1]->setLocalPos(
351         gRobotLink[1]->getGlobalTransform().getTranslation() );
352     meshCadLink[1]->setLocalRot(
353         gRobotLink[1]->getGlobalTransform().getRotation() );
354
355     // link 1 COM graphic update
```

```
356     sphereCenterOfMass[0]->setLocalPos( ...
            gRobotLink[1]->getCenterOfMass()->getGlobalTransform().getTranslation() ...
            );
357     sphereCenterOfMass[0]->setLocalRot( ...
            gRobotLink[1]->getCenterOfMass()->getGlobalTransform().getRotation() ...
            );
358
359     // link 2 cad graphic update
360     meshCadLink[2]->setLocalPos(
361         gRobotLink[2]->getGlobalTransform().getTranslation() );
362     meshCadLink[2]->setLocalRot(
363         gRobotLink[2]->getGlobalTransform().getRotation() );
364
365     // link 2 COM graphic update
366     sphereCenterOfMass[1]->setLocalPos(
367         gRobotLink[2]->getCenterOfMass()->
368         getGlobalTransform().getTranslation());
369     sphereCenterOfMass[1]->setLocalRot(
370         gRobotLink[2]->getCenterOfMass()->getGlobalTransform().getRotation());
371
372     // link 3 cad update
373     meshCadLink[3]->setLocalPos(
374         gRobotLink[3]->getGlobalTransform().getTranslation() );
375     meshCadLink[3]->setLocalRot(
376         gRobotLink[3]->getGlobalTransform().getRotation() );
377
378     // link 3 com graphic update
379     sphereCenterOfMass[2]->setLocalPos(
380         gRobotLink[3]->getCenterOfMass()->
```

```
381         getGlobalTransform().getTranslation() );
382     sphereCenterOfMass[2]->setLocalRot(
383         gRobotLink[3]->getCenterOfMass()->getGlobalTransform().getRotation() ...
                ;
384
385     // tool graphic update
386     for (int i = 0; i < 4; i++){
387         sphereTool[i]->setLocalPos(
388             gRobotTool[i]->getGlobalTransform().getTranslation() );
389         sphereTool[i]->setLocalRot(
390             gRobotTool[i]->getGlobalTransform().getRotation() );
391     }
392
393     // reference point update
394     sphereReference->setLocalPos(cVector3d(gHandleRefPos(0), ...
            gHandleRefPos(1), gHandleRefPos(2)));
395
396     // camera update
397     cameraTool->setLocalPos(
398         gRobotTool[1]->getGlobalTransform().getTranslation() );
399     cameraTool->setLocalRot(
400         gRobotTool[1]->getGlobalTransform().getRotation() );
401
402
403
404     //-----------------------------------------------------------
405     // UPDATE WIDGETS
406     //-----------------------------------------------------------
407
```

```
408     // update the force display scope
409     signalScope->setSignalValues(gHandleRefFrc(0), gHandleRefFrc(1),
410         gHandleFbkFrc(0), gHandleFbkFrc(1));
411
412     // display haptic rate data
413     labelHapticRate->setText(cStr(gFrequencyCounter.getFrequency(), 0) + " ...
            Hz");
414
415     // display the test mode
416     if (gOperatingMode == OPERATING_MODE_AUTONOMOUS) {
417         labelOperatingMode->setText("Test mode: AUTONOMOUS");
418     } else if (gOperatingMode == OPERATING_MODE_TELEOPERATION) {
419         labelOperatingMode->setText("Test mode: TELEOPERATION");
420     } else if (gOperatingMode == OPERATING_MODE_COLLABORATION) {
421         labelOperatingMode->setText("Test mode: COLLABORATION");
422     }
423
424     // display the test status
425     if (gTestRunning) {
426         labelTestStatus->setText("Test Status: RUNNING");
427     } else {
428         labelTestStatus->setText("Test Status: NOT RUNNING");
429     }
430
431     // update position of label
432     labelHapticRate->setLocalPos((int)(0.5 *
433         (windowW - labelHapticRate->getWidth())), 15);
434
435     // update velocity information to level
```

```
    double val = gAutonomy->getWork() / gAutonomy->m_workTarget;
    levelVelocity->setValue(val);

    if (val ≥1) {
        levelVelocity->m_colorActive = cColorf(0.05, 0.8, 0.05);
    } else {
        levelVelocity->m_colorActive = cColorf(0.8, 0.05, 0.05);
    }

    // update the coupon transparency with localization confidence
    meshCadCoupon->setTransparencyLevel(gLocalizationConf);

    // update the image
    if (gCameraConnected) {
        if (gImageLock.tryAcquire()) {
            imageDataPtr->loadFromFile("frame.jpg");
            imageBitmap->loadFromImage(imageDataPtr);
            imageBitmap->setTransparencyLevel(gCameraTransparency);
            imageBitmap->setUseTransparency(true);
            gImageLock.release();
        }
    }

    //-----------------------------------------------------------
    // RENDER SCENE
    //-----------------------------------------------------------

    // update shadow maps (if any)
    // gHapticWorld->updateShadowMaps(false, mirroredDisplay);
```

```
465
466     // render side framebuffer
467     frameBuffer->renderView();
468
469     // render world
470     camera->renderView(windowW, windowH);
471
472     // swap buffers
473     glutSwapBuffers();
474
475     // wait until all GL commands are completed
476     glFinish();
477 }
478
479 // callback when the window display is resized
480 void resizeWindow(int w, int h)
481 {
482     windowW = w;
483     windowH = h;
484
485     // update size of framebuffer and view panel
486     int scale = 0.4 * w;
487     // int scale = 0.6 * cMin(w, h);
488     int sw = 1.0 * scale;
489     int sh = (9.0 / 16.0) * scale;
490     frameBuffer->setSize(sw, sh);
491
492     // update the view panel
493     int radius = 0.01 * scale;
```

```
494     viewPanel->setSize(sw, sh);
495     viewPanel->setCornerRadius(radius, radius, radius, radius);
496
497     // update size of signalScope
498     signalScope->setSize(sw, 0.7 * sh);
499     signalScope->setLocalPos(w - 20 - sw, 20);
500     signalScope->setCornerRadius(radius, radius, radius, radius);
501
502     // update the size of levelVelocity
503     double widthScale = 0.05;
504     levelVelocity->setSize(widthScale * w);
505     levelVelocity->setLocalPos(20, 60 + sh);
506
507     // update size of messagePanel
508     messagePanel->setSize(sw, 0.2 * sh);
509     messagePanel->setLocalPos(w - 20 - sw, 20 + 0.8 * sh);
510     messagePanel->setCornerRadius(radius, radius, radius, radius);
511
512     // update position of labels
513     labelTestStatus->setLocalPos(20, h - 40);
514     labelOperatingMode->setLocalPos(20, h - 60);
515     labelMessage->setLocalPos(w + 20 - sw, 20 + 0.82 * sh);
516
517     // update the image
518     imageBitmap->setCornerRadius(radius, radius, radius, radius);
519     double zw = double(sw) / double(gCameraWidth);
520     double zh = double(sh) / double(gCameraHeight);
521     imageBitmap->setZoom(zw, zh);
522 }
```

```
523
524 // mouse click on image
525 void mouseClick(int button, int state, int x, int y)
526 {
527     mouseX = x;
528     mouseY = y;
529 }
530
531 // mouse moves on image
532 void mouseMove(int x, int y)
533 {
534     // compute mouse motion
535     int dx = x - mouseX;
536     int dy = y - mouseY;
537     mouseX = x;
538     mouseY = y;
539
540     // compute ne camera angles
541     double azimuthDeg = camera->getSphericalAzimuthDeg() + (0.5 * dy);
542     double polarDeg = camera->getSphericalPolarDeg() + (-0.5 * dx);
543
544     // assign new angles
545     camera->setSphericalAzimuthDeg(azimuthDeg);
546     camera->setSphericalPolarDeg(polarDeg);
547 }
548
549 // function that closes the application
550 void close(void)
551 {
```

```
552
553     // stop the simulation
554     gSimulationRunning = false;
555
556     // wait for graphics, haptics, and kinematics loops to terminate
557     // while (!gHapticsFinished) { cSleepMs(100); }
558     // while (!gRobotFinished) { cSleepMs(100); }
559
560     // close haptic device
561     gHapticDevice->close();
562 }
563
564 // callback of GLUT timer
565 void graphicsTimer(int data)
566 {
567     if (gSimulationRunning){
568         glutPostRedisplay();
569     }
570     // glutPostRedisplay();
571     glutTimerFunc(60, graphicsTimer, 0);
572 }
573
574 #endif
```

Listing C.15: src/main.cpp - Main program

```
1 /*
2  Copyright 2014-2019, Parker Owan
3  PhD Dissertation, University of Washington
```

```
 */

#include <algorithm>

// Include the headers
#include "main.h"

// Thread callbacks
#include "updateControl.h"
#include "updateSignals.h"
#include "updateAutonomy.h"
#include "updateVision.h"

// ****************************************************************
// main loop
// ****************************************************************
int main(int argc, char* argv[]) {

    std::cout << "\n-----------------------------\n\n";
    std::cout << "Hole cleaning application\n";
    std::cout << "(c) 2018 Parker Owan\n\n";
    std::cout << "-----------------------------\n";

    //----------------------------------------------------------
    // GET INPUT ARGUMENTS
    //----------------------------------------------------------
    for (int i = 0; i < argc; ++i) {
        std::string arg = argv[i];

```

```
33        if ((arg == "-h") || (arg == "--help")) {
34            std::cout << "Available input arguments:\n";
35            std::cout << "\t '-s'         - simulation only\n";
36            std::cout << "\t '-ins=<value> "
37                << "- instructions: 0 = hole 5, 1 = holes 1-9 (default)\n";
38            std::cout << "\t '-cam=<value> "
39                << "- camera index (default = 0)\n";
40            std::cout << "\t '-vra=<value> "
41                << "- camera/virtual overlay transparency (default = 1.0)\n";
42            std::cout << "\t '-wsr=<value> "
43                << "- haptic gain: workspace radius (default = 0.4).\n";
44            std::cout << "\t '-wsc=<value> "
45                << "- haptic gain: workspace center (default = 0.2).\n";
46            std::cout << "\t '-kpd=<value>' "
47                << "- haptic gain: stiffness (default = 650).\n";
48            std::cout << "\t '-kdd=<value>' "
49                << "- haptic gain: damping (default = 1).\n";
50            std::cout << "\t '-gam=<value>' "
51                << "- haptic gain: force reflection (default = 3.5).\n";
52            std::cout << "\t '-kph=<value>' "
53                << "- robot gain: handle stiffness (default = 1250).\n";
54            std::cout << "\t '-kdh=<value>' "
55                << "- robot gain: handle damping (default = 10).\n";
56            std::cout << "\t '-kpt=<value>' "
57                << "- robot gain: tip orientation stiffness (default = 10).\n";
58            std::cout << "\t '-ati=<string>' "
59                << "- ati IP address (default = 172.28.14.10).\n";
60            std::cout << "\n------------------------------\n";
61            return 0;
```

```
62          }
63
64          if (arg.substr(0, 2) == "-s") {
65              gSimulationOnly = true;
66              std::cout << "Set simulation only\n";
67          }
68
69          if (arg.substr(0, 4) == "-ins") {
70              std::string str = arg.substr(5, 20); // get the value
71              gInstructions = unsigned(atoi(str.c_str()));
72              std::cout << "ins = " << gInstructions << "\n";
73          }
74
75          if (arg.substr(0, 4) == "-cam") {
76              std::string str = arg.substr(5, 20); // get the value
77              gCameraIndex = unsigned(atoi(str.c_str()));
78              std::cout << "cam = " << gCameraIndex << "\n";
79          }
80
81          if (arg.substr(0, 4) == "-vra") {
82              std::string str = arg.substr(5, 20); // get the value
83              gCameraTransparency = atof(str.c_str());
84              std::cout << "vra = " << gCameraTransparency << "\n";
85          }
86
87          if (arg.substr(0, 4) == "-wsr") {
88              std::string str = arg.substr(5, 20); // get the value
89              gWorkspaceRadius = atof(str.c_str());
90              std::cout << "wsr = " << gWorkspaceRadius << "\n";
```

```
91          }
92
93          if (arg.substr(0, 4) == "-wsc") {
94              std::string str = arg.substr(5, 20); // get the value
95              gWorkspaceCenterX = atof(str.c_str());
96              std::cout << "wsc = " << gWorkspaceCenterX << "\n";
97          }
98
99          if (arg.substr(0, 4) == "-kpd") {
100             std::string str = arg.substr(5, 20); // get the value
101             gKpd = atof(str.c_str());
102             std::cout << "Kpd = " << gKpd << "\n";
103         }
104
105         if (arg.substr(0, 4) == "-kdd") {
106             std::string str = arg.substr(5, 20); // get the value
107             gKdd = atof(str.c_str());
108             std::cout << "Kdd = " << gKdd << "\n";
109         }
110
111         if (arg.substr(0, 4) == "-gam") {
112             std::string str = arg.substr(5, 20); // get the value
113             gGam = atof(str.c_str());
114             std::cout << "Gam = " << gGam << "\n";
115         }
116
117         if (arg.substr(0, 4) == "-kph") {
118             std::string str = arg.substr(5, 20); // get the value
119             gKph = atof(str.c_str());
```

```
120             std::cout << "Kph = " << gKph << "\n";
121         }
122
123         if (arg.substr(0, 4) == "-kdh") {
124             std::string str = arg.substr(5, 20); // get the value
125             gKdh = atof(str.c_str());
126             std::cout << "Kdh = " << gKdh << "\n";
127         }
128
129         if (arg.substr(0, 4) == "-kpt") {
130             std::string str = arg.substr(5, 20); // get the value
131             gKpt = atof(str.c_str());
132             std::cout << "Kpt = " << gKpt << "\n";
133         }
134
135         if (arg.substr(0, 4) == "-kpt") {
136             atiAddress = arg.substr(5, 20); // get the value
137             std::cout << "ATI IP = " << atiAddress << "\n";
138         }
139
140         std::cout << "\n";
141     }
142
143     // Command options
144     std::cout << "Key Command Options:\n";
145     std::cout << "\t 'q' - quit the application\n";
146     std::cout << "\t 'f' - toggle fullscreen\n";
147     std::cout << "\t 'k' - toggle bilateral teleoperation\n";
148     std::cout << "\t 't' - toggle test modes\n";
```

```
149     std::cout << "\t ' ' - toggle test start/stop\n";
150     std::cout << "------------------------------\n";
151
152     //------------------------------------------------------------
153     // OPENGL - WINDOW DISPLAY
154     //------------------------------------------------------------
155
156     // initialize GLUT
157     glutInit(&argc, argv);
158
159     // retrieve resolution of computer display and position window accordingly
160     screenW = glutGet(GLUT_SCREEN_WIDTH);
161     screenH = glutGet(GLUT_SCREEN_HEIGHT);
162     windowW = (int)(0.8 * screenH);
163     windowH = (int)(0.5 * screenH);
164     windowPosY = (screenH - windowH) / 2;
165     windowPosX = windowPosY;
166
167     // initialize the OpenGL GLUT window
168     glutInitWindowPosition(windowPosX, windowPosY);
169     glutInitWindowSize(windowW, windowH);
170
171     // create display context and initialize GLEW library
172     glutCreateWindow(argv[0]);
173
174 #ifdef GLEW_VERSION
175     // initialize GLEW
176     glewInit();
177 #endif
```

```
178
179     // setup GLUT options
180     glutDisplayFunc(updateGraphics);
181     glutKeyboardFunc(keySelect);
182     glutMouseFunc(mouseClick);
183     glutMotionFunc(mouseMove);
184     glutReshapeFunc(resizeWindow);
185     glutSetWindowTitle("Application Viewer");
186
187     // set fullscreen mode
188     if (gFullscreen)
189     {
190         glutFullScreen();
191     }
192
193     //--------------------------------------------------------------
194     // SETUP CHAI3D WORLD
195     //--------------------------------------------------------------
196
197     // Variables for consistency
198     double toolRadius = 0.0038;
199     double pointRadius = 0.0025;
200     double frameSize = 0.05;
201     cVector3d worldCenter(gWorkspaceCenterX, 0, 0.1);
202
203     // Init world
204     gHapticWorld = new cWorld();
205     gHapticWorld->m_backgroundColor.setBlack();
206
```

```
207     camera = new cCamera(gHapticWorld);
208     gHapticWorld->addChild(camera);
209
210     // create a background
211     chai3d::cBackground* background = new chai3d::cBackground();
212     camera->m_backLayer->addChild(background);
213
214     // set background properties
215     background->setCornerColors(chai3d::cColorf(0.9f, 0.9f, 0.9f),
216                                 chai3d::cColorf(0.9f, 0.9f, 0.9f),
217                                 chai3d::cColorf(0.52f, 0.53f, 0.59f),
218                                 chai3d::cColorf(0.52f, 0.53f, 0.59f));
219
220     // define a basis in spherical coordinates for the camera
221     camera->setSphericalReferences(worldCenter, // origin
222                                    cVector3d(0, 0, 1), // zenith direction
223                                    cVector3d(1, 0, 0)); // azimuth direction
224
225     // set the camera orientation
226     camera->setSphericalDeg(1.0, // spherical coordinate radius
227                             50,  // spherical coordinate azimuth angle
228                             180); // spherical coordinate polar angle
229
230     camera->setClippingPlanes(0.01, 10.0);
231     camera->setUseMultipassTransparency(true);
232     cDirectionalLight* light = new cDirectionalLight(gHapticWorld);
233     camera->addChild(light);
234     light->setEnabled(true);
235     light->setDir(-3.0, -0.5, 0.0);
```

```
236
237     //--------------------------------------------------------------
238     // UI WIDGETS
239     //--------------------------------------------------------------
240
241     // setup a camera framebuffer
242     cameraTool = new cCamera(gHapticWorld);
243     cameraTool->setClippingPlanes(0.005, 1.0);
244     cameraTool->m_backLayer->addChild(background);
245
246     // attach camera to world
247     gHapticWorld->addChild(cameraTool);
248
249     // create framebuffer for side view
250     frameBuffer = cFrameBuffer::create();
251     frameBuffer->setup(cameraTool);
252
253     // create panel to display side view
254     viewPanel = new cViewPanel(frameBuffer);
255     camera->m_frontLayer->addChild(viewPanel);
256     viewPanel->setLocalPos(20, 20);
257
258     // create panel to display side view
259     messagePanel = new cPanel;
260     camera->m_frontLayer->addChild(messagePanel);
261     messagePanel->setLocalPos(20, 20);
262     messagePanel->setTransparencyLevel(0.85);
263
264     // create and allocate a new image pointer
```

```
265     imageDataPtr = cImage::create();
266
267     // create an image bitmap
268     imageBitmap = new cBitmap();
269     imageBitmap->loadFromImage(imageDataPtr);
270     imageBitmap->setLocalPos(20, 20);
271     camera->m_frontLayer->addChild(imageBitmap);
272
273     // create a level to display velocity data
274     levelVelocity = new cLevel();
275     camera->m_frontLayer->addChild(levelVelocity);
276     levelVelocity->setLocalPos(20, 60);
277     levelVelocity->setRange(0.0, 1.0);
278     levelVelocity->setSize(80);
279     levelVelocity->setNumIncrements(60);
280     levelVelocity->setSingleIncrementDisplay(false);
281     levelVelocity->setTransparencyLevel(0.85);
282
283     // create a scope
284     signalScope = new cScope();
285     camera->m_frontLayer->addChild(signalScope);
286     signalScope->setLocalPos(400, 20);
287     signalScope->setTransparencyLevel(0.85);
288     signalScope->setRange(-20, 20); // forces
289     signalScope->m_colorSignal0 = cColorf(.6, 0, 0);
290     signalScope->m_colorSignal1 = cColorf(0, .6, 0);
291     signalScope->m_colorSignal2 = cColorf(1, 0, 0);
292     signalScope->m_colorSignal3 = cColorf(0, 1, 0);
293
```

```
294     // enable signals for display
295     signalScope->setSignalEnabled(true, true, true, true);
296
297     // create a font
298     cFont *font1 = NEW_CFONTCALIBRI20();
299     cFont *font2 = NEW_CFONTCALIBRI24();
300
301     // create a label to display the haptic rate of the simulation
302     labelHapticRate = new cLabel(font1);
303     camera->m_frontLayer->addChild(labelHapticRate);
304
305     // create a label to display the haptic rate of the simulation
306     labelMessage = new cLabel(font2);
307     camera->m_frontLayer->addChild(labelMessage);
308
309     // create a label to display the current test mode
310     labelOperatingMode = new cLabel(font1);
311     labelOperatingMode->m_fontColor = cColorf(0, 0, 0);
312     camera->m_frontLayer->addChild(labelOperatingMode);
313
314     // create a label to display the current test status
315     labelTestStatus = new cLabel(font1);
316     labelTestStatus->m_fontColor = cColorf(0, 0, 0);
317     camera->m_frontLayer->addChild(labelTestStatus);
318
319     //-----------------------------------------------------------
320     // HAPTIC DEVICE
321     //-----------------------------------------------------------
322
```

```
323     // Setup tool
324     cHapticDeviceHandler* handler = new cHapticDeviceHandler();
325     handler->getDevice(gHapticDevice, 0);
326     cHapticDeviceInfo info = gHapticDevice->getSpecifications();
327     gHapticDevice->setEnableGripperUserSwitch(true);
328     gHapticTool = new cToolCursor(gHapticWorld);
329     gHapticWorld->addChild(gHapticTool);
330     gHapticTool->setHapticDevice(gHapticDevice);
331     gHapticTool->setRadius(toolRadius);
332
333     // change rotation & position (to align with base reference frame)
334     gHapticTool->setLocalPos(worldCenter);
335     gHapticTool->setLocalRot(cMatrix3d(-1.0, 0.0, 0.0,
336                                0.0, -1.0, 0.0,
337                                0.0, 0.0, 1.0));
338
339     // map the haptic device workspace to a larger virtual workspace.
340     gHapticTool->setWorkspaceRadius(gWorkspaceRadius);
341     double workspaceScaleFactor = gHapticTool->getWorkspaceScaleFactor();
342     cHapticDeviceInfo hapticDeviceInfo = gHapticDevice->getSpecifications();
343     double maxStiffness =
344         hapticDeviceInfo.m_maxLinearStiffness / workspaceScaleFactor;
345
346     // print out the workspace scale factor
347     std::cout << "The workspace scale factor is: " <<
348         workspaceScaleFactor << "\n";
349
350     // create a white cursor & show the frame
351     gHapticTool->m_hapticPoint->m_sphereProxy->m_material->setWhite();
```

```
352     gHapticTool->m_hapticPoint->m_sphereProxy->setShowFrame(false);
353     gHapticTool->m_hapticPoint->m_sphereProxy->setFrameSize(frameSize);
354
355     // haptic forces are enabled only if small forces are first sent to the
356     // device; this mode avoids the force spike that occurs when the ...
            application
357     // starts when the tool is located inside an object for instance.
358     gHapticTool->setWaitForSmallForce(true);
359
360     // start the haptic tool
361     gHapticTool->start();
362
363     //--------------------------------------------------------------
364     // CORE ROBOTICS SIMULATION WORLD
365     //--------------------------------------------------------------
366
367     // Create a new simulation world
368     cr::world::OriginPtr simWorld = cr::world::Origin::create();
369     simWorld->setName("Simulation world");
370
371     // Create a frame for writing the data
372     cr::physics::FrameEuler fe;
373     fe.setMode(gEulerConvention);
374
375     // Setup the target item
376     gTargetItem = cr::world::Node::create();
377     fe.setPositionAndOrientation(gTargetX, gTargetY, gTargetZ, 0, 0, 0);
378     gTargetItem->setLocalTransform(fe);
379     gTargetItem->setName("Target Location");
```

```
380     simWorld->addChild(gTargetItem);
381
382     // Setup the world items
383     gWorldItem[0] = cr::world::Node::create();
384     fe.setPositionAndOrientation(gHoleOffsetFromTargetX, 0, 0, 0, 0, 0);
385     gWorldItem[0]->setLocalTransform(fe);
386     gWorldItem[0]->setName("Test coupon");
387     gTargetItem->addChild(gWorldItem[0]);
388
389     // update the fiducial items
390     for (int i = 0; i < 9; i++){
391         std::string str = "Hole fiducial ";
392         str.append(std::to_string(i));
393         gWorldItem[i+1] = cr::world::Node::create();
394         fe.setPositionAndOrientation(gHoleFiducials[i][0],
395             gHoleFiducials[i][1], 0, 0, 0, 0);
396         gWorldItem[i+1]->setLocalTransform(fe);
397         gWorldItem[i+1]->setName(str);
398         gWorldItem[0]->addChild(gWorldItem[i+1]);
399     }
400
401     // Create the manipulator item
402     gRobot = cr::world::Robot::create();
403     gRobot->setName("My robot");
404     simWorld->addChild(gRobot);
405
406     // Define an inertia tensor matrix and com vector
407     Eigen::Matrix3d I;
408     Eigen::Vector3d COM;
```

```
409
410     // Create rigid body for the links
411     cr::physics::RigidBody rb;
412
413     // Setup the rigid body link items
414     gRobotLink[0] = cr::world::Link::create();
415     fe.setPositionAndOrientation(0, 0, 0.08415, 0, 0, 0);
416     gRobotLink[0]->setLocalTransform(fe);
417     gRobotLink[0]->setName("Robot base");
418     gRobot->addChild(gRobotLink[0]);
419
420     gRobotLink[1] = cr::world::Link::create();
421     fe.setPositionAndOrientation(0, 0, 0, 0, 0, - 3.0 * M_PI / 4.0);
422     I << 0, 0, 0, 0, 0, 0, 0, 0, 1870e-6;
423     COM << 0.136, 0, 0.045;
424     rb.setMass(0.68);
425     rb.setInertiaTensor(I);
426     rb.setCenterOfMass(COM);
427     gRobotLink[1]->setLocalTransform(fe);
428     gRobotLink[1]->setRigidBody(rb);
429     gRobotLink[1]->setDegreeOfFreedom(cr::physics::CR_EULER_FREE_ANG_G);
430     gRobotLink[1]->setName("Robot Link 0");
431     gRobotLink[0]->addChild(gRobotLink[1]);
432
433     gRobotLink[2] = cr::world::Link::create();
434     fe.setPositionAndOrientation(0.16905, 0, 0.07105, 0, 0, 2.0 * M_PI / 4.0);
435     I << 0, 0, 0, 0, 0, 0, 0, 0, 1870e-6;
436     COM << 0.136, 0, 0.045;
437     rb.setMass(0.68);
```

```
438     rb.setInertiaTensor(I);
439     rb.setCenterOfMass(COM);
440     gRobotLink[2]->setLocalTransform(fe);
441     gRobotLink[2]->setRigidBody(rb);
442     gRobotLink[2]->setDegreeOfFreedom(cr::physics::CR_EULER_FREE_ANG_G);
443     gRobotLink[2]->setName("Robot Link 1");
444     gRobotLink[1]->addChild(gRobotLink[2]);
445
446     gRobotLink[3] = cr::world::Link::create();
447     fe.setPositionAndOrientation(0.18008, 0, 0.07105, 0, 0, 1.0 * M_PI / 4.0);
448     I << 0, 0, 0, 0, 0, 0, 0, 0, 1200e-6;
449     COM << 0.014, 0, 0.040;
450     rb.setMass(0.375);
451     rb.setInertiaTensor(I);
452     rb.setCenterOfMass(COM);
453     gRobotLink[3]->setLocalTransform(fe);
454     gRobotLink[3]->setRigidBody(rb);
455     gRobotLink[3]->setDegreeOfFreedom(cr::physics::CR_EULER_FREE_ANG_G);
456     gRobotLink[3]->setName("Robot Link 2");
457     gRobotLink[2]->addChild(gRobotLink[3]);
458
459
460     // Setup tools and sensors that are attached to the robot
461     gRobotTool[0] = cr::world::Node::create();
462     fe.setPositionAndOrientation(0.1552, 0, 0.032, 0, 0, 0);
463     gRobotTool[0]->setLocalTransform(fe);
464     gRobotTool[0]->setName("Tool tip");
465     gRobotLink[3]->addChild(gRobotTool[0]);
466
```

```
467     gRobotTool[1] = cr::world::Node::create();
468     fe.setPositionAndOrientation(0.04337, 0, 0.068527, 0, M_PI / 18.0, M_PI);
469     gRobotTool[1]->setLocalTransform(fe);
470     gRobotTool[1]->setName("Camera sensor");
471     gRobotLink[3]->addChild(gRobotTool[1]);
472
473     gRobotTool[2] = cr::world::Node::create();
474     fe.setPositionAndOrientation(0.04370, 0, 0.032, 0, M_PI / 2.0, M_PI);
475     // fe.setPositionAndOrientation(0.04370, 0, 0.032, 0, 0, 0);
476     gRobotTool[2]->setLocalTransform(fe);
477     gRobotTool[2]->setName("ATI force sensor");
478     gRobotLink[3]->addChild(gRobotTool[2]);
479
480     gRobotTool[3] = cr::world::Node::create();
481     fe.setPositionAndOrientation(-0.0700, 0, 0.032, 0, 0, 0);
482     gRobotTool[3]->setLocalTransform(fe);
483     gRobotTool[3]->setName("Tool handle");
484     gRobotLink[3]->addChild(gRobotTool[3]);
485
486     // Important! Now we need to register the links with the Manipulator
487     // so we can compute Jacobians and masses, etc...
488     gRobot->addLink(gRobotLink[0]);
489     gRobot->addLink(gRobotLink[1]);
490     gRobot->addLink(gRobotLink[2]);
491     gRobot->addLink(gRobotLink[3]);
492
493     // Display the created world list
494     simWorld->print(std::cout);
495
```

```
496     //-----------------------------------------------------------
497     // CHAI3D WORLD & CAD ROBOT
498     //-----------------------------------------------------------
499
500     // init a pos offset vector
501     Eigen::Vector3d pos;
502
503     // define the stifnesses
504     double stiffness = std::min(0.6 * maxStiffness, 400.0);
505
506     // coupon mesh
507     meshCadCoupon = new cMultiMesh();
508     gHapticWorld->addChild(meshCadCoupon);
509     meshCadCoupon->loadFromFile("../../resources/mascot_bench.obj");
510     // meshCadCoupon->createAABBCollisionDetector(toolRadius);
511     // meshCadCoupon->setShowCollisionDetector(showCollisionDetector, true);
512     meshCadCoupon->setStiffness(stiffness, true);
513     meshCadCoupon->setFriction(0.1, 0.2, true);
514     meshCadCoupon->setUseDisplayList(true);
515     meshCadCoupon->setUseCulling(false);
516     meshCadCoupon->setTransparencyLevel(0.75);
517     meshCadCoupon->setEnabled(true);
518
519     // create the robot multimesh
520     meshCadLink[0] = new cMultiMesh();
521     gHapticWorld->addChild(meshCadLink[0]);
522     meshCadLink[0]->loadFromFile("../../resources/mascot_base.obj");
523     meshCadLink[0]->setUseDisplayList(true);
524
```

```
525     meshCadLink[1] = new cMultiMesh();
526     gHapticWorld->addChild(meshCadLink[1]);
527     meshCadLink[1]->loadFromFile("../../resources/mascot_link1.obj");
528     meshCadLink[1]->setUseDisplayList(true);
529
530     meshCadLink[2] = new cMultiMesh();
531     gHapticWorld->addChild(meshCadLink[2]);
532     meshCadLink[2]->loadFromFile("../../resources/mascot_link2.obj");
533     meshCadLink[2]->setUseDisplayList(true);
534
535     meshCadLink[3] = new cMultiMesh();
536     gHapticWorld->addChild(meshCadLink[3]);
537     meshCadLink[3]->loadFromFile("../../resources/Tool.obj");
538     meshCadLink[3]->setUseDisplayList(true);
539
540     // Robot tool point
541     for (int i = 0; i < 4; i++){
542         sphereTool[i] = new cShapeSphere(toolRadius);
543         sphereTool[i]->m_material->setRedCrimson();
544         sphereTool[i]->setShowFrame(true);
545         sphereTool[i]->setFrameSize(frameSize);
546         gHapticWorld->addChild(sphereTool[i]);
547     }
548
549     // Robot center of mass
550     for (int i = 0; i < 3; i++){
551         sphereCenterOfMass[i] = new cShapeSphere(toolRadius);
552         sphereCenterOfMass[i]->m_material->setGray();
553         sphereCenterOfMass[i]->setShowFrame(true);
```

```
554         sphereCenterOfMass[i]->setFrameSize(frameSize);
555         gHapticWorld->addChild(sphereCenterOfMass[i]);
556     }
557
558     // Create the hole location fiducial indicators
559     for (int i = 0; i < 9; i++){
560         sphereFiducial[i] = new cShapeSphere(pointRadius);
561         sphereFiducial[i]->m_material->setGreenLime();
562         sphereFiducial[i]->setTransparencyLevel(0.75);
563         sphereFiducial[i]->setShowFrame(false);
564         sphereFiducial[i]->setFrameSize(0.6 * frameSize);
565         gHapticWorld->addChild(sphereFiducial[i]);
566         boxFiducial[i] = new cShapeBox(0.1, 0.025, 0.025);
567         boxFiducial[i]->m_material->setBlueSky();
568         boxFiducial[i]->setTransparencyLevel(0.35);
569         boxFiducial[i]->setShowFrame(false);
570         boxFiducial[i]->setFrameSize(0.6 * frameSize);
571         gHapticWorld->addChild(boxFiducial[i]);
572     }
573
574     // reference point
575     sphereReference = new cShapeSphere(1.2 * pointRadius);
576     sphereReference->m_material->setBlack();
577     gHapticWorld->addChild(sphereReference);
578
579     // create an offset for the multisegment item
580     cShapeSphere* tgOffset = new cShapeSphere(0.6 * pointRadius);
581     tgOffset->m_material->setBlueCyan();
582     tgOffset->setLocalPos(cVector3d(gHoleOffsetFromTargetX, 0, 0));
```

```
583     gHapticWorld->addChild(tgOffset);
584
585     // create line segments for trajectories
586     trajectorySegment = new cMultiSegment();
587     tgOffset->addChild(trajectorySegment);
588     trajectorySegment->setUseDisplayList(true);
589
590     // set up particle display
591     cMultiPoint* cloud = new cMultiPoint();
592     gHapticWorld->addChild(cloud);
593     cr::noise::NoiseUniform rng;
594     Eigen::VectorXd xa(3);
595     Eigen::VectorXd xb(3);
596     xa << +0.35, -0.05, 0.24;
597     xb << +0.45, 0.05, 0.28;
598     rng.setParameters(xa, xb);
599     cColorf color;
600     color.setGraySlate();
601     unsigned num_particles = 2000;
602     for (unsigned i = 0; i < num_particles; i ++) {
603         xa = rng.sample();
604         cloud->newPoint(cVector3d(xa(0), xa(1), xa(2)), color);
605     }
606     cloud->setPointSize(2.0);
607     cloud->setUseDisplayList(true);
608     cloud->setEnabled(false);
609
610     //-----------------------------------------------------------
611     // SET THE INITIAL CONDIITONS FOR THE GLOBAL SIGNALS
```

```
612     //-------------------------------------------------------------
613
614     // initialize the global signal vectors
615     gTargetLocation << gTargetX, gTargetY, gTargetZ, 0, 0, 0;
616     gTargetEstimate = gTargetLocation;
617     gTargetVariance << gTargetVarX, gTargetVarY, 0, 0, 0, 0;
618     gJointCmdTrq.setZero(NUM_MODULES);
619     gJointFbkPos.setZero(NUM_MODULES);
620     gJointFbkVel.setZero(NUM_MODULES);
621     gJointFbkTrq.setZero(NUM_MODULES);
622     gAtiFbkFrc.setZero(6);
623     gHandleRefPos.setZero(6);
624     gHandleRefVel.setZero(6);
625     gHandleRefFrc.setZero(6);
626     gHandleCmdFrc.setZero(6);
627     gHumnRefFrc.setZero(6);
628     gHandleFbkPos.setZero(6);
629     gHandleFbkVel.setZero(6);
630     gHandleFbkFrc.setZero(6);
631
632     // get the default positions
633     gJointFbkPos = gRobot->getConfiguration();
634
635     // compute the kinematic relationships
636     gHandleFbkPos = gRobotTool[3]->
637         getGlobalTransform().getPose(gEulerConvention);
638     gHandleRefPos = gHandleFbkPos;
639
640     //-------------------------------------------------------------
```

```
641     // SET UP AUTONOMY
642     //------------------------------------------------------------
643
644     // Set up the autonomy object
645     gAutonomy = new AutonomyStep(trajectorySegment,
646         labelMessage, gHoleFiducials);
647
648     //------------------------------------------------------------
649     // START THE THREAD LOOPS
650     //------------------------------------------------------------
651
652     // print
653     std::cout << "Starting the program\n";
654
655     // start the global timer
656     gTimer.startTimer();
657
658     // set the running flag high
659     gSimulationRunning = true;
660
661     // create and start the robot control thread
662     cr::core::Thread robotThread =
663         cr::core::Thread(cr::core::CR_PRIORITY_HIGHEST);
664     robotThread.setCallback(*updateControl);
665
666     // create and start the signal logging thread
667     cr::core::Thread signalThread =
668         cr::core::Thread(cr::core::CR_PRIORITY_LOWEST);
669     signalThread.setCallback(*updateSignals);
```

```
670
671     // start the autonomy loop
672     cr::core::Thread autonomyThread =
673         cr::core::Thread(cr::core::CR_PRIORITY_NORMAL);
674     autonomyThread.setCallback(*updateAutonomy);
675
676     // start the vision loop
677     cr::core::Thread visionThread =
678         cr::core::Thread(cr::core::CR_PRIORITY_LOWEST);
679     visionThread.setCallback(*updateVision);
680
681     // setup callback when application exits
682     atexit(close);
683
684     // print
685     std::cout << "Starting graphics thread\n";
686
687     // start the main graphics rendering loop
688     glutTimerFunc(30, graphicsTimer, 0);
689     glutMainLoop();
690
691     return 0;
692 }
```

Listing C.16: src/updateControl.h - Bilateral control thread callback function

```
1 /*
2  Copyright 2014-2019, Parker Owan
3  PhD Dissertation, University of Washington
```

```
4  */
5
6  #ifndef UPDATE_CONTROL_H_
7  #define UPDATE_CONTROL_H_
8
9  #include "main.h"
10
11 // ************************************************************
12 // Jacobian of the ATI (this is needed since something is up with
13 // the CoreRobotics Jacobian--specifically the rotation matrix.)
14 // ************************************************************
15 Eigen::MatrixXd atiJacobian(Eigen::VectorXd q) {
16     double l1 = 0.16905; // (m) link 1
17     double l2 = 0.18008; // (m) link 2
18     double la = 0.04370; // (m) force/torque sensor
19     Eigen::MatrixXd J(6,3);
20     J << 0,
21     0,
22     0,
23     -l1*cos(q(0)) - l2*cos(q(0)+q(1)) - la*cos(q(0)+q(1)+q(2)),
24     -l2*cos(q(0)+q(1)) - la*cos(q(0)+q(1)+q(2)),
25     -la*cos(q(0)+q(1)+q(2)),
26     -l1*sin(q(0)) - l2*sin(q(0)+q(1)) - la*sin(q(0)+q(1)+q(2)),
27     -l2*sin(q(0)+q(1)) - la*sin(q(0)+q(1)+q(2)),
28     -la*sin(q(0)+q(1)+q(2)),
29     1,
30     1,
31     1,
32     0,
```

```
33      0,
34      0,
35      0,
36      0,
37      0;
38      return J;
39  }
40
41  // ***************************************************************
42  // Bilateral force-control loop
43  // ***************************************************************
44  void updateControl(void) {
45
46      // desired loop rate
47      double dt_robot = 0.001; // seconds
48
49      // Start a timer
50      cr::core::Clock timer;
51
52      // gainAlpha
53      double gainAlpha = 0;
54
55      //-----------------------------------------------------------
56      // HEBI SETUP
57      //-----------------------------------------------------------
58
59      // Create a command object; this can be sent to the group
60      hebi::GroupCommand command(NUM_MODULES);
61
```

```
62    // Create a group feedback object; will be filled in during the request.
63    hebi::GroupFeedback feedback(NUM_MODULES);
64
65    // create a hebi lookup object
66    hebi::Lookup lookup;
67
68    // create a vector of hebi macs
69    std::vector<hebi::MacAddress> macs;
70
71    // add mac addresses (for specific actuators)
72    macs.emplace_back(
73        hebi::MacAddress::fromBytes(0xd8, 0x80, 0x39, 0xef, 0x3b, 0xf0));
74    macs.emplace_back(
75        hebi::MacAddress::fromBytes(0xd8, 0x80, 0x39, 0xe8, 0x68, 0xb8));
76    macs.emplace_back(
77        hebi::MacAddress::fromBytes(0xd8, 0x80, 0x39, 0x9a, 0xc9, 0xaf));
78
79    // create a hebi group
80    std::shared_ptr<hebi::Group> group;
81
82    // connect to hebis
83    if (!gSimulationOnly) {
84
85        // creating hebi group
86        std::cout << "Checking for hebi connection...\n";
87
88        // lookup group
89        group = lookup.getGroupFromMacs(macs, 1000);
90
```

```
91        if (!group) {
92            printf("No group found\n");
93            gRobotConnected = false;
94            // return -1;
95        }
96        else {
97            if (group->size() != NUM_MODULES) {
98                printf("Group size was %i, we expcted %i!\n",
99                    group->size(), NUM_MODULES);
100               gRobotConnected = false;
101               // return -1;
102           }
103           else {
104               printf("Robot found\n");
105
106               // send command lifetime (0.1 s)
107               group->setCommandLifetimeMs(10);
108
109               // If we select this option, then we do not need to call a
110               // feedback request for every loop. This works out to be faster
111               // than the explicit feedback request call. Note this value
112               // might be lower on windows devices (800 Hz?)
113               group->setFeedbackFrequencyHz(1000); // Hz
114
115               // set the flag
116               gRobotConnected = true;
117
118               // get feedback from HEBI
119               // group->sendFeedbackRequest();
```

```
120             group->getNextFeedback(&feedback, 10);
121
122             // push feedback into vectors
123             gJointFbkPos = feedback.getPosition();
124
125             // push the feedback data to the robot representation
126             gRobot->setConfiguration(gJointFbkPos);
127
128             // compute the kinematic relationships
129             gHandleFbkPos = gRobotTool[3]->
130                 getGlobalTransform().getPose(gEulerConvention);
131             gHandleRefPos = gHandleFbkPos;
132         }
133     }
134 }
135
136 //-------------------------------------------------------------
137 // Simulation setup
138 //-------------------------------------------------------------
139
140 // setup the dynamic step class
141 DynamicsStep simDynamics(dt_robot);
142
143 // send it the joint angles
144 simDynamics.setPosition(gJointFbkPos);
145 simDynamics.setVelocity(gJointFbkVel);
146 simDynamics.setCommand(gJointCmdTrq);
147
148 //-------------------------------------------------------------
```

```
149     // ATI Setup
150     //-------------------------------------------------------------
151
152     // initialize the context
153     boost::asio::io_context ioContext;
154
155     // create an ati object
156     AtiFeedback atiSensor(ioContext);
157
158     // connect to the ATI
159     if (!gSimulationOnly) {
160
161         // print
162         std::cout << "Setting up ATI force/torque sensor...\n";
163
164         // connect
165         atiSensor.connect(atiAddress);
166
167         // apply bias - this doesn't do anything
168         // atiSensor.applyBias();
169     }
170
171     //-------------------------------------------------------------
172     // HAPTIC SETUP
173     //-------------------------------------------------------------
174
175     // handle Kd
176     Eigen::DiagonalMatrix<double, 3> gain_Kp;
177     gain_Kp.diagonal() << gKpd, gKpd, gKpd;
```

```
178
179     // tip Kp
180     Eigen::DiagonalMatrix<double, 3> gain_Kd;
181     gain_Kd.diagonal() << gKdd, gKdd, gKdd;
182
183     // frequency counter
184     chai3d::cFrequencyCounter frequencyCounter;
185
186     // gainBeta
187     double gainBeta = 0;
188
189     // reflection gain
190     double reflectionGain = gGam;
191
192     //-----------------------------------------------------------
193     // gains
194     //-----------------------------------------------------------
195
196     // handle Kp
197     Eigen::DiagonalMatrix<double, 6> gain_Kph;
198     gain_Kph.diagonal() << gKph, gKph, 0, 0, 0, 0;
199
200     // handle Kd
201     Eigen::DiagonalMatrix<double, 6> gain_Kdh;
202     gain_Kdh.diagonal() << gKdh, gKdh, 0, 0, 0, 0;
203
204     // tip Kp
205     Eigen::DiagonalMatrix<double, 6> gain_Kpt;
206     gain_Kpt.diagonal() << 0, 0, 0, 0, 0, gKpt;
```

```
207
208     //------------------------------------------------------------
209     // Loop controls
210     //------------------------------------------------------------
211
212     // variable for keeping track of elapsed time
213     double et = 0.0;
214
215     // wait a bit
216     timer.startTimer();
217     timer.sleep(0.1);
218
219     // set the gControlActive flag
220     gControlActive = true;
221
222     //------------------------------------------------------------
223     // main loop
224     //------------------------------------------------------------
225     std::cout << "Starting kinematics thread\n";
226     while (gSimulationRunning) {
227
228         // start timer
229         timer.startTimer();
230
231         //--------------------------------------------------------
232         // Feedback
233         //--------------------------------------------------------
234         if (gRobotConnected && !gSimulationOnly) {
235
```

```
236             // query ATI feedback
237             atiSensor.step();
238             gAtiFbkFrc = atiSensor.getFbkEffort();
239 
240             // get feedback from HEBI
241             // group->sendFeedbackRequest();
242             group->getNextFeedback(&feedback, 10);
243 
244             // push feedback into vectors
245             gJointFbkPos = feedback.getPosition();
246             gJointFbkVel = feedback.getVelocity();
247             gJointFbkTrq = feedback.getEffort();
248 
249         } else {
250 
251             // sim the robot motion
252             simDynamics.step();
253 
254             // get the feedback
255             gJointFbkPos = simDynamics.getPosition();
256             gJointFbkVel = simDynamics.getVelocity();
257         }
258 
259         //------------------------------------------------------
260         // KINEMATICS UPDATES
261         //------------------------------------------------------
262 
263         // push the feedback data to the robot representation
264         gRobot->setConfiguration(gJointFbkPos);
```

```
265
266         // tip jacobian
267         Eigen::MatrixXd Jt = gRobot->jacobian(gRobotTool[0], gEulerConvention);
268
269         // handle jacobian
270         Eigen::MatrixXd Jh = gRobot->jacobian(gRobotTool[3], gEulerConvention);
271
272         // force sensor jacobian - the CoreRobotics one doesn't seem to work
273         // for this version, so we do it manually. ATI is gRobotTool[2]
274         Eigen::MatrixXd Ja = atiJacobian(gJointFbkPos);
275
276         // compute the ATI force
277         Eigen::MatrixXd Jhh = Jh.transpose() * Jh;
278         gHandleFbkFrc = Jh * Jhh.inverse() * Ja.transpose() * gAtiFbkFrc;
279
280         // compute the kinematic relationships
281         Eigen::VectorXd xt = gRobotTool[0]->
282             getGlobalTransform().getPose(gEulerConvention);
283         gHandleFbkPos = gRobotTool[3]->
284             getGlobalTransform().getPose(gEulerConvention);
285         gHandleFbkVel = Jh * gJointFbkVel;
286
287         //-----------------------------------------------------
288         // HAPTICS
289         //-----------------------------------------------------
290
291         // compute global reference frames for each object
292         gHapticWorld->computeGlobalPositions(true);
293
```

```
294         // update position and orientation of tool
295         gHapticTool->updateFromDevice();
296 
297         // compute interaction forces
298         gHapticTool->computeInteractionForces();
299 
300         // get the haptic device position & velocity
301         cVector3d xd = gHapticTool->getDeviceGlobalPos();
302         cVector3d vd = gHapticTool->getDeviceGlobalLinVel();
303 
304         // Check if button pressed
305         gHapticDevice->getUserSwitch(0, gButtonPress);
306 
307         // if flag high, use bilateral teleop
308         Eigen::Vector3d u = gain_Kp * (gHandleFbkPos.topRows(3) - xd.eigen())
309             + gain_Kd * (gHandleFbkVel.topRows(3) - vd.eigen());
310         if (gBilateralEnabled)
311         {
312             gainBeta = (0.9995) * gainBeta + (0.0005) * 1.0;
313         }
314         else
315         {
316             gainBeta = (0.99) * gainBeta + (0.01) * 0.0;
317         }
318         u = gainBeta * u;
319         gHumnRefFrc.topRows(3) = - reflectionGain * u;
320 
321         // add the forces on
322         gHapticTool->addDeviceGlobalForce(u(0), u(1), u(2));
```

```
323
324         // send forces to haptic device
325         gHapticTool->applyToDevice();
326
327         //------------------------------------------------------
328         // BILATERAL CONTROL
329         //------------------------------------------------------
330
331         // tip controller
332         Eigen::VectorXd utip = gain_Kpt * ( -xt );
333
334         // handle controller
335         gHandleCmdFrc = gain_Kph * (gHandleRefPos - gHandleFbkPos) +
336             gain_Kdh * (gHandleRefVel - gHandleFbkVel) + gHandleRefFrc;
337
338         // compute the joint commands
339         Eigen::VectorXd uh = (gLevelOfAutonomy) * gHandleCmdFrc +
340             (1 - gLevelOfAutonomy) * gHumnRefFrc;
341         gJointCmdTrq = gainAlpha *
342             (Jt.transpose() * utip + Jh.transpose() * uh);
343
344         //------------------------------------------------------
345         // SET COMMANDS
346         //------------------------------------------------------
347         if (gRobotConnected && !gSimulationOnly) {
348
349             // apply to the hebi acutators
350             command.setEffort(gJointCmdTrq);
351             group->sendCommand(command);
```

```
352         } else {
353 
354             // send the command to the simulation
355             simDynamics.setCommand(gJointCmdTrq);
356         }
357 
358         //------------------------------------------------------
359         // LOOP
360         //------------------------------------------------------
361 
362         // ramp up gain alpha
363         gainAlpha = (0.9995) * gainAlpha + (0.0005) * 1;
364 
365         // compute the comm frequency and update
366         et = timer.getElapsedTime();
367 
368         // spinlock
369         if ((dt_robot - et) > 0) {
370             timer.sleep(dt_robot - et);
371         }
372 
373         // update frequency counter
374         gFrequencyCounter.signal(1);
375     }
376 }
377 
378 #endif
```

Listing C.17: src/updateAutonomy.h - Shared autonomy thread callback function

```
/*
 Copyright 2014-2019, Parker Owan
 PhD Dissertation, University of Washington
 */

#ifndef UPDATE_AUTONOMY_H_
#define UPDATE_AUTONOMY_H_

#include "main.h"

// ****************************************************************
// Autonomy update loop
// ****************************************************************
void updateAutonomy(void) {

    // desired shared memory update rate (200 Hz?)
    double dt_autonomy = 0.005; // seconds

    // loa filter parameter
    double fc_alpha = 1.0; // (Hz) cutoff frequency
    double tau_alpha = 1 / (2 * M_PI * fc_alpha); // (s) time constant
    double alpha = dt_autonomy / (dt_autonomy + tau_alpha); // loa smoothing

    // create the instructions - list of holes to clean
    std::vector<int> instructions;
    if (gInstructions == 0) {

        // clean hole 5
```

```
29          instructions.push_back(4);
30
31      } else if (gInstructions == 1) {
32
33          // clean holes 1-9
34          for (int i = 0; i < 9; i++) {
35              instructions.push_back(i);
36          }
37      }
38
39      // Start a timer
40      cr::core::Clock timer;
41
42      // variable for keeping track of elapsed time
43      double et = 0.0;
44
45      // set the autonomy sample rate
46      gAutonomy->setSampleRate(dt_autonomy);
47
48      // start the timer
49      timer.startTimer();
50
51      // wait until the control is active
52      while (!gControlActive) {
53          timer.sleep(0.2);
54      }
55
56      // initialize the autonomy object
57      Eigen::VectorXd xInitial = gHandleFbkPos;
```

```
58      gAutonomy->initialize(instructions, xInitial, gOperatingMode);
59
60
61      //-------------------------------------------------------------
62      // loop
63      //-------------------------------------------------------------
64      std::cout << "Starting autonomy thread\n";
65      while (gSimulationRunning) {
66
67          // start timer
68          timer.startTimer();
69
70          // level of autonomy (initialize so no teleoperation unless test)
71          double loa = 1.0;
72
73          // do stuff ...
74          if (gTestRunning) {
75
76              // set the button press
77              gAutonomy->setButton(gButtonPress);
78
79              // update the localization information
80              gAutonomy->setLocalization(gLocalizationConf,
81                                         gTargetEstimate);
82
83              // set the feedback to gAutonomy
84              gAutonomy->setFeedback(gHandleFbkPos,
85                                     gHandleFbkVel,
86                                     gHandleFbkFrc);
```

```
87
88              // step the autonomy
89              gAutonomy->step();
90
91              // get the level of autonomy
92              loa = gAutonomy->getLevelOfAutonomy();
93
94              // Set the reference commands
95              gAutonomy->getReference(gHandleRefPos,
96                                      gHandleRefVel,
97                                      gHandleRefFrc);
98
99              // stop the test
100             if ( gAutonomy->isDone() ) {
101                 gTestRunning = false;
102             }
103         }
104         else {
105
106             // initialize
107             gAutonomy->initialize(instructions, xInitial, gOperatingMode);
108         }
109
110         // filter the level of autonomy
111         gLevelOfAutonomy = (1-alpha) * gLevelOfAutonomy + (alpha) * loa;
112
113         // compute the comm frequency and update
114         et = timer.getElapsedTime();
115
```

```
116         // spinlock
117         if ((dt_autonomy - et) > 0) {
118             timer.sleep(dt_autonomy - et);
119         }
120     }
121 
122 }
123 
124 #endif
```

Listing C.18: src/updateVision.h - Vision thread callback function

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5  
6  #ifndef UPDATE_VISION_H_
7  #define UPDATE_VISION_H_
8  
9  #include "main.h"
10 
11 // ****************************************************************
12 // COMPUTE VISION LOOP
13 // ****************************************************************
14 void updateVision(void) {
15 
16     // Start a timer
17     cr::core::Clock timer;
```

```
18
19      // variable for keeping track of elapsed time
20      double et = 0.0;
21
22      // start the timer
23      timer.startTimer();
24
25      // wait until the control loop is active
26      while (!gControlActive) {
27          timer.sleep(0.6);
28      }
29
30      //--------------------------------------------------------
31      // opencv objects
32      //--------------------------------------------------------
33
34      // gray image
35      cv::Mat gray;
36
37      // video capture object & open
38      cv::VideoCapture cap;
39
40      // try to open the video object
41      cap.open(gCameraIndex);
42
43      // Camera properties (for consistency of test)
44      cap.set(CV_CAP_PROP_FRAME_WIDTH, gCameraWidth);
45      cap.set(CV_CAP_PROP_FRAME_HEIGHT, gCameraHeight);
46
```

```
47    // check is the video capture object was opened
48    if (cap.isOpened()) {
49
50        // read one frame
51        cap.read(gImageFrame);
52
53        // if it wasn't empyt, update the pointer
54        if (!gImageFrame.empty()) {
55
56            // save the image to png
57            if (gImageLock.acquire()) {
58                cv::imwrite("frame.jpg", gImageFrame);
59                gImageLock.release();
60            }
61
62            // set connected flag high
63            gCameraConnected = true;
64
65            // update the imageDataPtr to the cv::matrix
66            int width = gImageFrame.cols;
67            int height = gImageFrame.rows;
68            std::cout << "Webcam opened with size = " << width << " x " << ...
                  height << "\n";
69        }
70
71    } else {
72
73        // print
74        std::cout << "Webcam not opened.\n";
```

```
75      }
76
77
78      //-------------------------------------------------------------
79      // loop
80      //-------------------------------------------------------------
81      if (gCameraConnected) {
82
83          // only bother running this if the camera is connected
84          std::cout << "Starting vision thread\n";
85          while (gSimulationRunning) {
86
87              // start timer
88              timer.startTimer();
89
90              // if the camera is connected...
91              if (gCameraConnected) {
92
93                  // read the image
94                  cap.read(gImageFrame);
95                  if (!gImageFrame.empty()) {
96
97                      // save the image to png
98                      if (gImageLock.acquire()) {
99                          cv::imwrite("frame.jpg", gImageFrame);
100                         gImageLock.release();
101                     }
102
103                     // wait
```

```
104                     timer.sleep(0.05);
105
106                     // show the image
107                     #ifndef MACOSX
108                     cv::imshow("Computer Vision Image", gImageFrame);
109                     cv::waitKey(10);
110                     #endif
111                 }
112             }
113
114             // compute the comm frequency and update
115             et = timer.getElapsedTime();
116         }
117     }
118
119 }
120
121 #endif
```

Listing C.19: src/updateSignals.h - Signal logging thread callback function

```
1 /*
2  Copyright 2014-2019, Parker Owan
3  PhD Dissertation, University of Washington
4  */
5
6 #ifndef UPDATE_SIGNALS_H_
7 #define UPDATE_SIGNALS_H_
8
```

```
9  #include "main.h"
10
11 // ************************************************************
12 // Signal logging helpers
13 // ************************************************************
14
15 // write a double to the logfile
16 void logWriteNewline(std::ofstream& log) {
17     log << "\n";
18 }
19
20 // write a header string to the log file, where n is the signal size.
21 // when n > 0, the signal anem is appended with [%i]
22 // when n = 0, the signal name is not appended
23 void logWriteSignalHeader(std::ofstream& log, std::string str, int n) {
24     if (n > 0) {
25         for (int i = 0; i < n; i++) {
26             log << str << "[" << i << "],";
27         }
28     } else {
29         log << str << ",";
30     }
31 }
32
33 // write a string to the logfile (n times)
34 void logWriteString(std::ofstream& log, std::string str) {
35     log << str << ",";
36 }
37
```

```
38 // write a double to the logfile
39 void logWriteDouble(std::ofstream& log, double x) {
40     log << x << ",";
41 }
42
43 // write a vector to the logfile
44 void logWriteVector(std::ofstream& log, Eigen::VectorXd v) {
45     for (int k = 0; k < v.size(); k++) {
46         logWriteDouble(log, v(k));
47     }
48 }
49
50 // write a boolean to the logfile
51 void logWriteBoolean(std::ofstream& log, bool x) {
52     log << x << ",";
53 }
54
55 // write a boolean to the logfile
56 void logWriteInteger(std::ofstream& log, int x) {
57     log << x << ",";
58 }
59
60 // ************************************************************
61 // Signal logging loop
62 // ************************************************************
63 void updateSignals(void) {
64
65     // desired shared memory update rate (100 Hz?)
66     double dt_signals = 0.01; // seconds
```

```
67
68      // Start a timer
69      cr::core::Clock timer;
70
71      // variable for keeping track of elapsed time
72      double et = 0.0;
73
74      // start the timer
75      timer.startTimer();
76
77      // wait until the control is active
78      while (!gControlActive) {
79          timer.sleep(0.4);
80      }
81
82      //--------------------------------------------------------------
83      // Signals
84      //--------------------------------------------------------------
85      bool           sig_testRunning;
86      int            sig_actionIndex;
87      Eigen::VectorXd sig_holeIndex(1);
88      int            sig_sequenceIndex;
89      int            sig_automationState;
90      bool           sig_automationEnabled;
91      bool           sig_teleopEnabled;
92      bool           sig_teleopButton;
93      double         sig_levelOfAutonomy;
94      Eigen::VectorXd sig_resetEstimator(1);
95      Eigen::VectorXd sig_targetLocation(2);
```

```
96    Eigen::VectorXd sig_targetEstimate(2);
97    Eigen::VectorXd sig_targetVariance(2);
98    Eigen::VectorXd sig_targetConfidence(1);
99    Eigen::VectorXd sig_handleFrcFilt(2);
100   Eigen::VectorXd sig_handleFrcMu(2);
101   Eigen::VectorXd sig_handleFrcVar(2);
102   double          sig_nominalConf;
103   double          sig_nominalLik;
104   Eigen::VectorXd sig_jointFbkPos(3);
105   Eigen::VectorXd sig_jointFbkVel(3);
106   Eigen::VectorXd sig_jointFbkTrq(3);
107   Eigen::VectorXd sig_atiFbkFrc(6);
108   Eigen::VectorXd sig_handleRefPos(6);
109   Eigen::VectorXd sig_handleRefVel(6);
110   Eigen::VectorXd sig_handleRefFrc(6);
111   Eigen::VectorXd sig_handleCmdFrc(6);
112   Eigen::VectorXd sig_humnRefFrc(6);
113   Eigen::VectorXd sig_handleFbkPos(6);
114   Eigen::VectorXd sig_handleFbkVel(6);
115   Eigen::VectorXd sig_handleFbkFrc(6);
116
117   //-------------------------------------------------------------
118   // Write datalog header
119   //-------------------------------------------------------------
120   std::ofstream logFile;
121   logFile.open("datalog");
122   logWriteSignalHeader(logFile, "time", 0);
123   logWriteSignalHeader(logFile, "sig_testRunning", 1);
124   logWriteSignalHeader(logFile, "sig_actionIndex", 1);
```

```
125     logWriteSignalHeader(logFile, "sig_holeIndex", 1);
126     logWriteSignalHeader(logFile, "sig_sequenceIndex", 1);
127     logWriteSignalHeader(logFile, "sig_automationState", 1);
128     logWriteSignalHeader(logFile, "sig_automationEnabled", 1);
129     logWriteSignalHeader(logFile, "sig_teleopEnabled", 1);
130     logWriteSignalHeader(logFile, "sig_teleopButton", 1);
131     logWriteSignalHeader(logFile, "sig_levelOfAutonomy", 1);
132     logWriteSignalHeader(logFile, "sig_resetEstimator", 1);
133     logWriteSignalHeader(logFile, "sig_targetLocation", 2);
134     logWriteSignalHeader(logFile, "sig_targetEstimate", 2);
135     logWriteSignalHeader(logFile, "sig_targetVariance", 2);
136     logWriteSignalHeader(logFile, "sig_targetConfidence", 1);
137     logWriteSignalHeader(logFile, "sig_handleFrcFilt", 2);
138     logWriteSignalHeader(logFile, "sig_handleFrcMu", 2);
139     logWriteSignalHeader(logFile, "sig_handleFrcVar", 2);
140     logWriteSignalHeader(logFile, "sig_nominalConf", 1);
141     logWriteSignalHeader(logFile, "sig_nominalLik", 1);
142     logWriteSignalHeader(logFile, "sig_jointFbkPos", 3);
143     logWriteSignalHeader(logFile, "sig_jointFbkVel", 3);
144     logWriteSignalHeader(logFile, "sig_jointFbkTrq", 3);
145     logWriteSignalHeader(logFile, "sig_atiFbkFrc", 6);
146     logWriteSignalHeader(logFile, "sig_handleRefPos", 6);
147     logWriteSignalHeader(logFile, "sig_handleRefVel", 6);
148     logWriteSignalHeader(logFile, "sig_handleRefFrc", 6);
149     logWriteSignalHeader(logFile, "sig_handleCmdFrc", 6);
150     logWriteSignalHeader(logFile, "sig_humnRefFrc", 6);
151     logWriteSignalHeader(logFile, "sig_handleFbkPos", 6);
152     logWriteSignalHeader(logFile, "sig_handleFbkVel", 6);
153     logWriteSignalHeader(logFile, "sig_handleFbkFrc", 6);
```

```
154     logWriteNewline(logFile);
155
156     //-------------------------------------------------------------
157     // Shared Memory - for talking to MATLAB (localizeTargetOnline)
158     //-------------------------------------------------------------
159     cr::core::SharedMemory shm("mascot-memory", cr::core::CR_MANAGER_SERVER);
160
161     // Define the initial conditions for the signals
162     sig_holeIndex(0) = double(gAutonomy->getActiveHoleIndex());
163     sig_resetEstimator(0) = double(!gTestRunning);
164     sig_handleFbkPos = gHandleFbkPos;
165     sig_handleFbkVel = gHandleFbkVel;
166     sig_handleFbkFrc = gHandleFbkFrc;
167     sig_targetEstimate = gTargetEstimate.segment(0,2);
168     sig_targetVariance = gTargetVariance.segment(0,2);
169     sig_targetConfidence(0) = double(gLocalizationConf);
170
171     // add the signals to the shared memory
172     shm.addSignal("sig_holeIndex", sig_holeIndex);
173     shm.addSignal("sig_resetEstimator", sig_resetEstimator);
174     shm.addSignal("sig_handleFbkPos", sig_handleFbkPos);
175     shm.addSignal("sig_handleFbkVel", sig_handleFbkVel);
176     shm.addSignal("sig_handleFbkFrc", sig_handleFbkFrc);
177     shm.addSignal("sig_targetEstimate", sig_targetEstimate);
178     shm.addSignal("sig_targetVariance", sig_targetVariance);
179     shm.addSignal("sig_targetConfidence", sig_targetConfidence);
180
181     //-------------------------------------------------------------
182     // loop
```

```
183     //-------------------------------------------------------------
184     std::cout << "Starting signal thread\n";
185     while (gSimulationRunning) {
186
187         // start timer
188         timer.startTimer();
189
190         // copy assign the signals to set
191         sig_testRunning = gTestRunning;
192         sig_actionIndex = gAutonomy->m_knet->getActionIndex();
193         sig_holeIndex(0) = gAutonomy->getActiveHoleIndex();
194         sig_sequenceIndex = gAutonomy->getSequenceIndex();
195         sig_automationState = int(gAutonomy->getAutonomyState());
196         sig_automationEnabled = gAutonomy->getAutomationEnabled();
197         sig_teleopEnabled = gAutonomy->getTeleoperationEnabled();
198         sig_teleopButton = gButtonPress;
199         sig_levelOfAutonomy = gLevelOfAutonomy;
200         sig_resetEstimator(0) = double(!gTestRunning);
201         sig_targetLocation = gTargetLocation.segment(0,2);
202         sig_handleFrcFilt = gAutonomy->getHandleFrcFlt().segment(0,2);
203         sig_handleFrcMu = gAutonomy->getHandleFrcMu().segment(0,2);
204         sig_handleFrcVar = gAutonomy->getHandleFrcVar().segment(0,2);
205         sig_nominalConf = gAutonomy->getNominalConfidence();
206         sig_nominalLik = gAutonomy->getNominalLikelihood();
207         sig_jointFbkPos = gJointFbkPos;
208         sig_jointFbkVel = gJointFbkVel;
209         sig_jointFbkTrq = gJointFbkTrq;
210         sig_atiFbkFrc = gAtiFbkFrc;
211         sig_handleRefPos = gHandleRefPos;
```

```
212         sig_handleRefVel = gHandleRefVel;
213         sig_handleRefFrc = gHandleRefFrc;
214         sig_handleCmdFrc = gHandleCmdFrc;
215         sig_humnRefFrc = gHumnRefFrc;
216         sig_handleFbkPos = gHandleFbkPos;
217         sig_handleFbkVel = gHandleFbkVel;
218         sig_handleFbkFrc = gHandleFbkFrc;
219
220         // set feedback data to shared memory
221         shm.set("sig_holeIndex", sig_holeIndex);
222         shm.set("sig_resetEstimator", sig_resetEstimator);
223         shm.set("sig_handleFbkPos", sig_handleFbkPos);
224         shm.set("sig_handleFbkVel", sig_handleFbkVel);
225         shm.set("sig_handleFbkFrc", sig_handleFbkFrc);
226
227         // get reference data from shared memory
228         sig_targetEstimate = shm.get("sig_targetEstimate");
229         sig_targetVariance = shm.get("sig_targetVariance");
230         sig_targetConfidence = shm.get("sig_targetConfidence");
231
232         // copy assign the received values
233         gTargetEstimate.segment(0,2) = sig_targetEstimate;
234         gTargetVariance.segment(0,2) = sig_targetVariance;
235         gLocalizationConf = sig_targetConfidence(0);
236
237         // write to datalog
238         logWriteDouble(logFile, gTimer.getElapsedTime());
239         logWriteBoolean(logFile, sig_testRunning);
240         logWriteInteger(logFile, sig_actionIndex);
```

```
241         logWriteVector(logFile, sig_holeIndex);
242         logWriteInteger(logFile, sig_sequenceIndex);
243         logWriteInteger(logFile, sig_automationState);
244         logWriteBoolean(logFile, sig_automationEnabled);
245         logWriteBoolean(logFile, sig_teleopEnabled);
246         logWriteBoolean(logFile, sig_teleopButton);
247         logWriteDouble(logFile, sig_levelOfAutonomy);
248         logWriteVector(logFile, sig_resetEstimator);
249         logWriteVector(logFile, sig_targetLocation);
250         logWriteVector(logFile, sig_targetEstimate);
251         logWriteVector(logFile, sig_targetVariance);
252         logWriteVector(logFile, sig_targetConfidence);
253         logWriteVector(logFile, sig_handleFrcFilt);
254         logWriteVector(logFile, sig_handleFrcMu);
255         logWriteVector(logFile, sig_handleFrcVar);
256         logWriteDouble(logFile, sig_nominalConf);
257         logWriteDouble(logFile, sig_nominalLik);
258         logWriteVector(logFile, sig_jointFbkPos);
259         logWriteVector(logFile, sig_jointFbkVel);
260         logWriteVector(logFile, sig_jointFbkTrq);
261         logWriteVector(logFile, sig_atiFbkFrc);
262         logWriteVector(logFile, sig_handleRefPos);
263         logWriteVector(logFile, sig_handleRefVel);
264         logWriteVector(logFile, sig_handleRefFrc);
265         logWriteVector(logFile, sig_handleCmdFrc);
266         logWriteVector(logFile, sig_humnRefFrc);
267         logWriteVector(logFile, sig_handleFbkPos);
268         logWriteVector(logFile, sig_handleFbkVel);
269         logWriteVector(logFile, sig_handleFbkFrc);
```

```
270         logWriteNewline(logFile);
271
272         // compute the comm frequency and update
273         et = timer.getElapsedTime();
274
275         // spinlock
276         if ((dt_signals - et) > 0) {
277             timer.sleep(dt_signals - et);
278         }
279     }
280
281     // close the log file
282     logFile.close();
283
284 }
285
286 #endif
```

C.2.3 Automation library

Listing C.20: lib/AutonomyStep.h - Automation class header

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #ifndef AUTONOMY_STEP_H_
7  #define AUTONOMY_STEP_H_
8
9  #include <cr/core>
10 #include "chai3d.h"
11 #include "Knet.h"
12
13 // ---------------------------------------------------
14 // Enuemrator for defining the state of autonomy
15 // ---------------------------------------------------
16 enum AutonomyState {
17     AUTONOMY_WAITING,
18     AUTONOMY_OFF_NOMINAL_EVENT,
19     AUTONOMY_GENERATING_TRAJECTORY,
20     AUTONOMY_TRACKING_TRAJECTORY,
21     AUTONOMY_CLEANING,
22     HUMAN_CLEANING,
23     HUMAN_MOVEMENT,
24 };
25
```

```
26 // ------------------------------------------------
27 // Enuemrator for defining the mode of operation
28 // ------------------------------------------------
29 enum OperatingMode {
30     OPERATING_MODE_AUTONOMOUS,
31     OPERATING_MODE_TELEOPERATION,
32     OPERATING_MODE_COLLABORATION,
33 };
34
35 // ------------------------------------------------
36 // Autonomy step
37 // ------------------------------------------------
38 class AutonomyStep : public cr::core::Step
39 {
40
41     // constructor and destructor
42     public:
43         AutonomyStep(chai3d::cMultiSegment* segment,
44                      chai3d::cLabel* label,
45                      double targets[9][2]);
46         ¬AutonomyStep();
47
48     // primary step element (this is where all the logic happens)
49     public:
50         virtual void step();
51
52     // subroutines
53     public:
54         bool checkNominal();
```

```
55      void initialize(std::vector<int>& instructions,
56                      Eigen::VectorXd& xhInitial,
57                      OperatingMode mode);
58      void planTrajectory(Eigen::VectorXd& xh,
59                          Eigen::VectorXd& vh);
60      void getTrajectory(chai3d::cMultiSegment* trajectorySegment);
61
62  // get/set
63  public:
64      void setSampleRate(double dt) { m_dt = dt; }
65      bool isDone() { return m_taskComplete; }
66      void setButton(bool button) { m_button = button; }
67      void setFeedback(Eigen::VectorXd position,
68                       Eigen::VectorXd velocity,
69                       Eigen::VectorXd force);
70      void getReference(Eigen::VectorXd& position,
71                        Eigen::VectorXd& velocity,
72                        Eigen::VectorXd& force);
73      void setLocalization(double conf, Eigen::VectorXd center);
74      void setSequenceIndex(int index) { m_sequenceIndex = index; }
75      int getSequenceIndex() { return m_sequenceIndex; }
76      int getAutonomyState() { return m_automationState; }
77      double getLevelOfAutonomy() { return m_loa; }
78      double getWork() { return m_work; }
79      int getActiveHoleIndex();
80      unsigned getHoleIndex();
81      bool getAutomationEnabled() { return (m_loa == 1.0); }
82      bool getTeleoperationEnabled() { return (m_loa == 0.0); }
83      Eigen::VectorXd getHandleFrcFlt() { return m_fltFrc; }
```

```
84          Eigen::VectorXd getHandleFrcMu() { return m_fltExp; }
85          Eigen::VectorXd getHandleFrcVar() { return m_fltVar; }
86          double getNominalConfidence() { return m_nominalConfidence; }
87          double getNominalLikelihood() { return m_nominalLikelihood; }
88
89      // knet
90      public:
91
92          //! Knet
93          Knet* m_knet;
94
95          //! Target work (J)
96          const double m_workTarget = 6.0;
97
98          //! Critical nominal confidence threshold
99          const double m_nominalCritical = 0.003;
100
101         //! Critical localization threshold
102         const double m_localizationCritical = 0.997;
103
104         //! Critical mahalanobis distance before advancing to the cleaning task
105         // const double m_distanceCritical = 1.0;
106         const double m_distanceCritical = 20.0;
107
108     // properties
109     private:
110
111         //! clock
112         cr::core::Clock m_timer;
```

```
113
114         //! sample rate (s)
115         double m_dt = 0.005;
116
117         //! done flag
118         bool m_taskComplete = false;
119
120         //! End time (s)
121         double m_tend = 0;
122
123         //! Accumulated work
124         double m_work = 0;
125
126         //! Current automation state
127         AutonomyState m_automationState = AUTONOMY_WAITING;
128
129         //! Operating mode
130         OperatingMode m_operatingMode = OPERATING_MODE_AUTONOMOUS;
131
132         //! level of autonomy
133         double m_loa = 1;
134
135         //! Sequence index flag
136         int m_sequenceIndex = 0;
137
138         //! Nominal confidence interval
139         double m_nominalConfidence = 1;
140
141         //! Nominal likelihood interval
```

```
142         double m_nominalLikelihood = 1;
143
144         //! Localization confidence interval
145         double m_localizationConfidence = 1;
146
147         //! Localization estimate
148         // Eigen::VectorXd m_centerEstimate;
149
150         //! button press state
151         bool m_button = false;
152
153         //! previous button press state
154         bool m_ButtonPrev = false;
155
156         //! CHAI3D Trajectory segment
157         chai3d::cMultiSegment* m_segment;
158
159         //! CHAI3D text label
160         chai3d::cLabel* m_label;
161
162         //! Instruction list (pair holes to sequence)
163         std::vector<int> m_instructions;
164
165         //! List of target locations
166         std::vector<Eigen::VectorXd> m_targets;
167
168         //! Localization reference (adopted when confidence)
169         Eigen::VectorXd m_centerPoint;
170
```

```
171         //! Initial position
172         Eigen::VectorXd m_handleInitial;
173
174         //! Brush cleaning initial offset (from global target location)
175         Eigen::VectorXd m_x0;
176
177         //! sigma inverse (for proximity to hole starting position)
178         Eigen::MatrixXd m_sig0Inv;
179
180         //! trajectory generator object
181         cr::control::TrajectoryGenerator m_tg;
182
183         //! force measurement covariance
184         Eigen::MatrixXd m_Rf;
185
186         //! Reference states
187         Eigen::VectorXd m_refPos;
188         Eigen::VectorXd m_refVel;
189         Eigen::VectorXd m_refFrc;
190
191         //! Feedback states
192         Eigen::VectorXd m_fbkPos;
193         Eigen::VectorXd m_fbkVel;
194         Eigen::VectorXd m_fbkFrc;
195
196         //! Force expectation states
197         Eigen::VectorXd m_expFrc;
198         Eigen::MatrixXd m_varFrc;
199
```

```
200         //! Filter states
201         Eigen::VectorXd m_fltFrc; // measured force
202         Eigen::VectorXd m_fltExp; // expeccted force
203         Eigen::MatrixXd m_fltVar; // expected covariance
204 };
205
206 #endif
```

Listing C.21: lib/AutonomyStep.cpp - Automation class source

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #include "AutonomyStep.h"
7
8  #include <algorithm>  // std::max
9  #include <math.h>     // sqrt
10
11 // ------------------------------------------------
12 // Constructor
13 // ------------------------------------------------
14 AutonomyStep::AutonomyStep(chai3d::cMultiSegment* segment,
15                            chai3d::cLabel* label,
16                            double targets[9][2]) {
17
18     // Initialize the Knet
19     m_knet = new Knet();
```

```
20
21      // Define the center point
22      m_centerPoint.setZero(6);
23      // m_centerEstimate.setZero(6);
24      // m_centerPoint(0) = 0.1859;
25      // m_centerPoint(1) = 0.0146;
26      // m_centerPoint(2) = 0.25825;
27
28      // Compute the target offset locations
29      Eigen::VectorXd xd(6);
30      xd.setZero(6);
31      for (int i = 0; i < 9; i++) {
32          xd(0) = targets[i][0];
33          xd(1) = targets[i][1];
34          m_targets.push_back(xd);
35      }
36
37      // cleaning initial offset value
38      // These values come from the KNET - first point in the clustering
39      m_x0.setZero(6);
40      m_x0(0) = -0.036431995048368;
41      m_x0(1) = -0.001603676276638;
42
43      // sigma inverse (from KNET - first cluster covariance)
44      Eigen::MatrixXd sig0;
45      sig0.setZero(4,4);
46      sig0 << 0.1158e-4,  0.0002e-4,  -0.0042e-4,  -0.0020e-4,
47              0.0002e-4,   0.1038e-4,  -0.0013e-4,   0.0005e-4,
48             -0.0042e-4,  -0.0013e-4,   0.1549e-4,   0.0081e-4,
```

```
49          -0.0020e-4,   0.0005e-4,   0.0081e-4,   0.1349e-4;
50      m_sig0Inv = sig0.inverse();
51
52      // set the force covariance
53      m_Rf.setZero(6,6);
54      m_Rf.diagonal() << 30, 20, 0, 0, 0, 0;
55      // m_Rf.diagonal() << 23, 15, 0, 0, 0, 0;
56
57      // initialize the reference members
58      m_refPos.setZero(6);
59      m_refVel.setZero(6);
60      m_refFrc.setZero(6);
61
62      // initialize the feedback members
63      m_fbkPos.setZero(6);
64      m_fbkVel.setZero(6);
65      m_fbkFrc.setZero(6);
66
67      // initialize the filtered feedback members
68      m_fltFrc.setZero(6);
69      m_fltExp.setZero(6);
70      m_fltVar = m_Rf;
71
72      // initialize the expectation members
73      m_expFrc.setZero(6);
74      m_varFrc.setZero(6,6);
75
76      // get the trajectory segement
77      m_segment = segment;
```

```
78
79      // get the text label
80      m_label = label;
81  }
82
83  // ------------------------------------------------
84  // Destructor
85  // ------------------------------------------------
86  AutonomyStep::¬AutonomyStep()
87  {
88      delete m_segment;
89      delete m_knet;
90  }
91
92  // ------------------------------------------------
93  // Step (autonomy state machine)
94  // ------------------------------------------------
95  void AutonomyStep::step()
96  {
97      // Set the nominal expected forces
98      m_expFrc.setZero(6);
99      m_varFrc = m_Rf;
100
101     // null the reference forces
102     m_refFrc.setZero(6);
103
104     // get time
105     double t = m_timer.getElapsedTime();
106
```

```
107     // if the task is not done...
108     if (!m_taskComplete) {
109
110         // MODE: human movement
111         if (m_automationState == HUMAN_MOVEMENT) {
112
113             // if the button was just pressed...
114             if (m_button && !m_ButtonPrev) {
115
116                 // udpate the automation state
117                 m_automationState = HUMAN_CLEANING;
118
119                 // set the label
120                 m_label->setText("It looks like you started cleaning.");
121             }
122         }
123
124         // MODE: human cleaning
125         if (m_automationState == HUMAN_CLEANING) {
126
127             // compute the work
128             double power = - m_fbkVel(0) * m_fbkFrc(0);
129             m_work += power * m_dt;
130
131             // tell them that the hole is clean if the work is sufficient
132             if (m_work ≥m_workTarget) {
133
134                 // set the label
135                 m_label->setText("The hole is clean, time to move on.");
```

```
136
137                 // if the button was just released...
138                 if (!m_button && m_ButtonPrev) {
139
140                     // update the sequence index
141                     m_sequenceIndex++;
142
143                     // update the state
144                     m_automationState = HUMAN_MOVEMENT;
145
146                     // set the work to zero
147                     m_work = 0;
148
149                     // if we're in collaboration mode...
150                     if (m_operatingMode == OPERATING_MODE_COLLABORATION) {
151
152                         // If we're sufficiently confident in localization...
153                         if (m_localizationConfidence ≥m_localizationCritical)
154                         {
155
156                             // set the loa active
157                             m_loa = 1;
158
159                             // set the label
160                             m_label->setText(
161                                 "I'm taking over, your work here is done!");
162
163                             // update the automation state
164                             m_automationState = AUTONOMY_GENERATING_TRAJECTORY;
```

```
165                     } else {
166
167                         // set the label
168                         m_label->setText(
169                             "I'm not sure where the hole is yet, \
170                             please clean the next one for me.");
171                     }
172                 }
173             }
174         }
175     }
176
177     // MODE: trajectory plan needed
178     else if (m_automationState == AUTONOMY_GENERATING_TRAJECTORY) {
179
180         // generate a trajectory
181         AutonomyStep::planTrajectory(m_fbkPos, m_fbkVel);
182
183         // update the visualizer
184         getTrajectory(m_segment);
185
186         // udpate the automation state
187         m_automationState = AUTONOMY_TRACKING_TRAJECTORY;
188
189         // set the label
190         m_label->setText("I'm performing a trajectory.");
191
192         // restart the timer
193         m_timer.startTimer();
```

```
194         }
195
196         // MODE: trajectory tracking enabled
197         else if (m_automationState == AUTONOMY_TRACKING_TRAJECTORY) {
198
199             // if the trajectory is still active, compute it...
200             if (t < m_tend) {
201                 cr::control::Waypoint wp = m_tg.step(t);
202                 m_refPos = wp.position;
203                 m_refVel = wp.velocity;
204                 m_refFrc.setZero(6);
205             }
206             // otherwise, let's advance to the next state
207             else {
208
209                 // compute the mahal distance to the target
210                 Eigen::VectorXd e;
211                 e.setZero(4);
212                 e.segment(0, 2) = m_refPos.segment(0, 2) -
213                     m_fbkPos.segment(0, 2);
214                 e.segment(2, 2) = - m_fbkVel.segment(0, 2);
215                 double dist = e.transpose() * m_sig0Inv * e;
216
217                 // Wait for the system to get close to the target state
218                 if (dist < m_distanceCritical) {
219
220                     // initialize the knet
221                     m_knet->initialize();
222
```

```
223                 // start the knet timer
224                 m_knet->start();
225
226                 // restart the autonomy timer
227                 m_timer.startTimer();
228
229                 // udpate the automation state
230                 m_automationState = AUTONOMY_CLEANING;
231
232                 // set the label
233                 m_label->setText(
234                     "I'm performing the hole cleaning procedure.");
235             }
236         }
237     }
238
239     // MODE: brush cleaning
240     else if (m_automationState == AUTONOMY_CLEANING) {
241
242         // If we're still in the task...
243         if (m_sequenceIndex < m_instructions.size()) {
244
245             // compute the work
246             double power = - m_fbkVel(0) * m_fbkFrc(0);
247             m_work += power * m_dt;
248             m_knet->setWork(m_work);
249
250             // step the knet
251             m_knet->step();
```

```
252
253             // Get the position and velocity from the knet
254             int holeIndex = m_instructions.at(m_sequenceIndex);
255
256             m_refPos = m_targets.at(holeIndex) + m_centerPoint;
257             Eigen::VectorXd x = m_knet->getPosition(); // these are dim 2
258             Eigen::VectorXd v = m_knet->getVelocity(); // these are dim 2
259             m_refPos(0) = x(0) + m_refPos(0);
260             m_refVel(0) = v(0);
261
262             // Get the feedforward force from reference
263             Eigen::VectorXd frcMu;
264             Eigen::MatrixXd frcSigma;
265             m_knet->predictForce(x, v, frcMu, frcSigma);
266             m_refFrc(0) = -frcMu(0);
267
268             // Get the expected force from feedback
269             Eigen::VectorXd expMu;
270             Eigen::MatrixXd expSigma;
271             Eigen::VectorXd xh = m_fbkPos -
272                 (m_targets.at(holeIndex) + m_centerPoint);
273             m_knet->predictForce(xh.segment(0,2),
274                 m_fbkVel.segment(0, 2), expMu, expSigma);
275             m_expFrc.segment(0, 2) = expMu;
276
277             // check if we're done cleaning
278             if (m_knet->isDone()) {
279
280                 // set the work to zero
```

```
281                     m_work = 0;
282
283                     //udpate the automation state
284                     m_automationState = AUTONOMY_GENERATING_TRAJECTORY;
285
286                     // advance the sequence index
287                     m_sequenceIndex++;
288                 }
289             }
290             // Otherwise, we're done...
291             else {
292
293                 // udpate the automation state
294                 m_automationState = AUTONOMY_WAITING;
295
296                 // set the task over flag
297                 m_taskComplete = true;
298
299                 // update the label
300                 m_label->setText("I'm done with the task.");
301             }
302         }
303     }
304
305     // check that the condition is nominal
306     if (!checkNominal()) {
307
308         // if the autonomy is active, then this is a problem...
309         if (m_loa == 1) {
```

```
310
311             // let's set the level of autonomy low
312             m_loa = 0;
313
314             // update message
315             m_label->setText("I encountered an off-nominal condition,\
316                 please complete this hole for me.");
317
318             // if we're in collaboration mode...
319             if (m_operatingMode == OPERATING_MODE_COLLABORATION) {
320
321                 // check if the automation was cleaning...
322                 if (m_automationState == AUTONOMY_CLEANING) {
323
324                     // no button press required, just start cleaning...
325                     m_automationState = HUMAN_CLEANING;
326                 } else {
327
328                     // set this to human movemenet
329                     m_automationState = HUMAN_MOVEMENT;
330                 }
331
332             // otherwise, let's enter a lockout state...
333             } else {
334
335                 // this requires a test restart to get back to nominal
336                 m_automationState = AUTONOMY_OFF_NOMINAL_EVENT;
337             }
338         }
```

```
339     }
340
341     // advance the button press state
342     m_ButtonPrev = m_button;
343 }
344
345 // ------------------------------------------------
346 // check if we're in nominal operation
347 // For this section, see Chapter 4.
348 // ------------------------------------------------
349 bool AutonomyStep::checkNominal()
350 {
351     // compute the transition matrix
352     double humanTau = 0.1; // (s)
353     double at = m_dt / humanTau;
354     Eigen::Matrix2d Amarkov;
355     Amarkov << 1-at, at, at, 1-at;
356
357     // compute the filter parameter
358     double fc_filt = 10; // (Hz) cutoff frequency
359     double tau_filt = 1 / (2 * M_PI * fc_filt); // (s) time constant
360     double a_filt = m_dt / (m_dt + tau_filt);;
361
362     // filter the values
363     m_fltFrc = (1-a_filt) * m_fltFrc + (a_filt) * m_fbkFrc;
364     m_fltExp = (1-a_filt) * m_fltExp + (a_filt) * m_expFrc;
365     m_fltVar = (1-a_filt) * m_fltVar + (a_filt) * m_varFrc;
366
367     // Check off-nominal
```

```
368     Eigen::Vector2d eta, lik;
369     eta << m_nominalConfidence, 1-m_nominalConfidence;
370     eta = Amarkov * eta;
371     double arg = pow(m_fltFrc(0) - m_fltExp(0), 2) / m_fltVar(0,0);
372     m_nominalLikelihood = exp(-arg / 2);
373     lik << m_nominalLikelihood, 1.0 - m_nominalLikelihood;
374     eta = eta.cwiseProduct(lik);
375     eta = eta * (1 / eta.sum());
376     m_nominalConfidence = eta(0);
377 
378     // now check if we tripped an off-nominal case
379     return (m_nominalConfidence ≥m_nominalCritical);
380 }
381 
382 // ------------------------------------------------
383 // Set the feedback values
384 // ------------------------------------------------
385 void AutonomyStep::setFeedback(Eigen::VectorXd position,
386                                Eigen::VectorXd velocity,
387                                Eigen::VectorXd force)
388 {
389     m_fbkPos = position;
390     m_fbkVel = velocity;
391     m_fbkFrc = force;
392 }
393 
394 // ------------------------------------------------
395 // Get the reference values
396 // ------------------------------------------------
```

```
397 void AutonomyStep::getReference(Eigen::VectorXd& position,
398                               Eigen::VectorXd& velocity,
399                               Eigen::VectorXd& force)
400 {
401     position = m_refPos;
402     velocity = m_refVel;
403     force = m_refFrc;
404 }
405
406 // ------------------------------------------------
407 // Initialize
408 // ------------------------------------------------
409 void AutonomyStep::initialize(std::vector<int>& instructions,
410                               Eigen::VectorXd& xhInitial,
411                               OperatingMode mode) {
412
413     // Define the nominal center point
414     m_centerPoint.setZero(6);
415     m_centerPoint(0) = 0.1859;
416     m_centerPoint(1) = 0.0146;
417     m_centerPoint(2) = 0.25825;
418
419     // reset the sequence index
420     m_sequenceIndex = 0;
421
422     // Setup the instruction list
423     m_instructions = instructions;
424
425     // Set the initial starting position
```

```
426     m_handleInitial = xhInitial;
427
428     // Set the reference state
429     m_refPos = xhInitial;
430     m_refVel.setZero(6);
431     m_refFrc.setZero(6);
432
433     // Set the operating mode
434     m_operatingMode = mode;
435
436     // Set the initial state
437     if (mode == OPERATING_MODE_AUTONOMOUS) {
438         m_automationState = AUTONOMY_GENERATING_TRAJECTORY;
439         m_loa = 1.0;
440     } else {
441         m_automationState = HUMAN_MOVEMENT;
442         m_loa = 0.0;
443     }
444
445     // Set the confidence interval
446     m_nominalConfidence = 1;
447     m_nominalLikelihood = 1;
448
449     // The task is not complete
450     m_taskComplete = false;
451
452     // Set work to zero
453     m_work = 0;
454
```

```
455     // message label
456     m_label->setText("Hello human.");
457 }
458 
459 // ------------------------------------------------
460 // Generate a min-jerk trajectory
461 // ------------------------------------------------
462 void AutonomyStep::planTrajectory(Eigen::VectorXd& xh,
463                                   Eigen::VectorXd& vh) {
464 
465     // create the needed parameters
466     Eigen::VectorXd x0(6);
467     Eigen::VectorXd v0(6);
468     Eigen::VectorXd a0(6);
469     Eigen::VectorXd x1(6);
470     Eigen::VectorXd v1(6);
471     Eigen::VectorXd a1(6);
472     m_tend = 2.0;
473 
474     // (m/s) the mean velocity parameter used for generating trajectories
475     double Vbar = 0.1;
476 
477     // if we finished the instructions, return home...
478     if (m_sequenceIndex ≥m_instructions.size()) {
479 
480         // set the WP states
481         x0 = xh;
482         v0 = vh;
483         a0 << 0, 0, 0, 0, 0, 0;
```

```
484         x1 = m_handleInitial;
485         v1.setZero(6);
486         a1 << +.5, 0, 0, 0, 0, 0;
487 
488         // compute the time
489         double d = (x1 - x0).norm();
490         m_tend = std::max(d / Vbar, 0.5);
491 
492         // message label
493         // m_label->setText("I'm returning home.");
494 
495     // otherwise, plan a trajectory to the next hole...
496     } else {
497 
498         // set the WP states
499         x0 = xh;
500         v0 = vh;
501         a0 << 0, 0, 0, 0, 0, 0;
502         int holeIndex = m_instructions.at(m_sequenceIndex);
503         x1 = m_targets.at(holeIndex) + m_centerPoint + m_x0;
504         v1.setZero(6);
505         a1 << .5, 0, 0, 0, 0, 0;
506 
507         // compute the time
508         double d = (x1 - x0).norm();
509         m_tend = std::max(d / Vbar, 0.5);
510 
511         // if we're near the hole, generate some nice motions between the holes
512         if (d < 0.08) {
```

```
513             a0(0) = -2.0;
514             a1(0) = -2.0;
515 
516         }
517         // otherwise, we'll just pull away from the hole before moving
518         else
519         {
520             a0(0) = 0.0;
521             a1(0) = std::max(-1.0 / m_tend, -3.0);
522         }
523 
524         // set the label
525         // m_label->setText("I'm performing a trajectory.");
526     }
527 
528     // solve the TG
529     m_tg.solve(x0, v0, a0, x1, v1, a1, m_tend);
530 }
531 
532 // ---------------------------------------------------
533 // Get the trajectory segments
534 // ---------------------------------------------------
535 void AutonomyStep::getTrajectory(
536     chai3d::cMultiSegment* trajectorySegment)
537 {
538 
539     // clear the trajectorySegment
540     trajectorySegment->clear();
541 
```

```
542     // loop
543     cr::control::Waypoint wp0, wp1;
544     Eigen::VectorXd x0;
545     Eigen::VectorXd x1;
546     double dt = 0.01;
547     double t = 0;
548     for (int i = 0; i < int(m_tend / dt)-1 ; i++) {
549         wp0 = m_tg.step(t);
550         wp1 = m_tg.step(t+dt);
551         x0 = wp0.position;
552         x1 = wp1.position;
553 
554         int index0 = trajectorySegment->newVertex(x0(0), x0(1), x0(2));
555         int index1 = trajectorySegment->newVertex(x1(0), x1(1), x1(2));
556 
557         // create segment by connecting both vertices together
558         trajectorySegment->newSegment(index0, index1);
559         t = t + dt;
560     }
561 
562     // set color
563     chai3d::cColorf color;
564     color.setGreenTeal();
565     trajectorySegment->setLineColor(color);
566     trajectorySegment->setLineWidth(4.0);
567 }
568 
569 // ---------------------------------------------------
570 // Set the localization update
```

```
571 // ------------------------------------------------
572 void AutonomyStep::setLocalization(double conf,
573                                    Eigen::VectorXd center)
574 {
575     m_localizationConfidence = conf;
576     m_centerPoint = center;
577 }
578 
579 // ------------------------------------------------
580 // Get the active hole index (returns -1 if not active)
581 // ------------------------------------------------
582 int AutonomyStep::getActiveHoleIndex() {
583     int holeIndex = -1;
584     if ((m_automationState == AUTONOMY_CLEANING) ||
585         (m_automationState == HUMAN_CLEANING))
586     {
587         if (m_sequenceIndex < m_instructions.size()) {
588             holeIndex = m_instructions.at(m_sequenceIndex);
589         }
590     }
591     return holeIndex;
592 }
593 
594 // ------------------------------------------------
595 // Get the hole index in the instruction set
596 // ------------------------------------------------
597 unsigned AutonomyStep::getHoleIndex() {
598     if (m_sequenceIndex < m_instructions.size()) {
599         return m_instructions.at(m_sequenceIndex);
```

```
600     } else {
601         if (m_instructions.size() > 0) {
602             return m_instructions.at(0);
603         } else {
604             return -1;
605         }
606     }
607 }
```

Listing C.22: lib/Knet.h - Cleaning policy class header

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #ifndef KNET_H_
7  #define KNET_H_
8
9  #include <cr/core>
10 #include <cr/control>
11 #include <cr/noise>
12 #include "Trajectory.h"
13
14 //-------------------------------------------------------------
15 // Transition Neural Network parameters
16 //-------------------------------------------------------------
17 struct NeuralParameters {
18     double         il_ymin;       // input layer minimum
```

```
19      Eigen::VectorXd il_xoffset;  // input layer offset [5 x 1]
20      Eigen::VectorXd il_gain;     // input layer gain [5 x 1]
21      Eigen::VectorXd l1_b;        // layer 1 bias [M x 1]
22      Eigen::MatrixXd l1_W;        // layer 1 weights [M x 5]
23      Eigen::VectorXd l2_b;        // layer 2 bias [4 x 1]
24      Eigen::MatrixXd l2_W;        // layer 2 weights [4 x M]
25  };
26
27  //-------------------------------------------------------------
28  // KNet Class
29  // To understand this, see the thesis Chapter 5.
30  //-------------------------------------------------------------
31  class Knet : public cr::core::Step
32  {
33
34      // constructor and destructor
35      public:
36          Knet();
37          ¬Knet();
38
39      // derived step elements
40      public:
41          void initialize ();
42          void start ();
43          void step ();
44
45      // routines
46      public:
47          bool isDone() { return m_done; }
```

```
48          Eigen::VectorXd getPosition() { return m_wp.position; }
49          Eigen::VectorXd getVelocity() { return m_wp.velocity; }
50          cr::control::Waypoint getWaypoint() { return m_wp; }
51          cr::noise::Gmm getAction(unsigned action) {
52              return m_actions.at(action); }
53          void setWork(double work) { m_work = work; }
54          int getActionIndex();
55
56      // prediction functions
57      public:
58
59          // predict force from position and velocity (for the current action)
60          void predictForce(Eigen::VectorXd x,
61                            Eigen::VectorXd v,
62                            Eigen::VectorXd& Mu,
63                            Eigen::MatrixXd& Sigma);
64
65          // predict accel from position and velocity (for the current action)
66          void predictAccel(Eigen::VectorXd x,
67                            Eigen::VectorXd v,
68                            Eigen::VectorXd& Mu,
69                            Eigen::MatrixXd& Sigma);
70
71      // neural network functions
72      public:
73
74          // transition neural network evaluation
75          Eigen::VectorXd transitionNetwork(Eigen::VectorXd u);
76
```

```
77      // properties
78      private:
79
80          // done flag
81          bool m_done = true;
82
83          // mechanical work done in task
84          double m_work = 0;
85
86          // time since last work computation
87          double m_t0 = 0;
88
89          // action index
90          unsigned m_action = 0;
91
92          // internal timer
93          cr::core::Clock m_timer;
94
95          // Trajectory representations
96          std::vector<Trajectory> m_trajectories;
97
98          // Gaussian mixture action representations
99          std::vector<cr::noise::Gmm> m_actions;
100
101         // Gaussian model representations
102         std::vector<cr::noise::NoiseGaussian> m_models;
103
104         // Current waypoint
105         cr::control::Waypoint m_wp;
```

```
106
107         // Neural network parameters
108         NeuralParameters m_nnet;
109 };
110
111 #endif
```

Listing C.23: lib/Knet.cpp - Cleaning policy class source

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #include "Knet.h"
7  #include "KnetConstants.h"
8
9  // ------------------------------------------------
10 // Constructor
11 // ------------------------------------------------
12 Knet::Knet()
13 {
14     // load the trajectories
15     // see: KnetConstans.h for loadKnetTrajectories()
16     Trajectory traj[KNET_ACTIONS];
17     loadKnetTrajectories(traj);
18     for (unsigned i = 0; i < KNET_ACTIONS; i++){
19         m_trajectories.push_back(traj[i]);
20     }
```

```
21
22      // load the actions
23      // see: KnetConstans.h for loadMixtureModels()
24      cr::noise::Gmm gmm[KNET_ACTIONS];
25      loadMixtureModels(gmm);
26      for (unsigned i = 0; i < KNET_ACTIONS; i++){
27          m_actions.push_back(gmm[i]);
28      }
29
30      // load the neural net parameters
31      // see: KnetConstans.h for loadNeuralParameters()
32      loadNeuralParameters(m_nnet);
33
34      // initialize
35      initialize();
36  }
37
38  // -------------------------------------------------
39  // Destructor
40  // -------------------------------------------------
41  Knet::¬Knet()
42  {
43      m_trajectories.clear();
44      m_actions.clear();
45  }
46
47  // -------------------------------------------------
48  // Initialize
49  // -------------------------------------------------
```

```
50 void Knet::initialize()
51 {
52     m_action = KNET_FIRST_ACTION; // starting action
53     m_done = false; // we're not done yet
54     Trajectory traj = m_trajectories.at(m_action);
55     cr::control::Waypoint wp = traj.step(0);
56     m_wp = wp;
57     m_work = 0;
58 }
59
60 // ------------------------------------------------
61 // Initialize
62 // ------------------------------------------------
63 void Knet::start()
64 {
65     m_timer.startTimer();
66     m_t0 = 0;
67 }
68
69 // ------------------------------------------------
70 // Step the knet
71 // ------------------------------------------------
72 void Knet::step()
73 {
74     if (!m_done) {
75
76         // get the ET
77         double t = m_timer.getElapsedTime();
78
```

```
79          // if the current trajectory is done, advance
80          if (m_trajectories.at(m_action).isDone(t)) {
81
82              // If we were in the last action
83              if (m_action == int(KNET_LAST_ACTION)) {
84
85                  // then we're done...
86                  m_done = true;
87
88                  // set the action index back to the first
89                  m_action = KNET_FIRST_ACTION;
90              }
91              // otherwise, let's figure out the next action
92              else {
93
94                  // setup an input/output vector
95                  Eigen::VectorXd u(5); // input vector [1 work + 4 actions]
96                  u.setZero(5);
97                  u(0) = m_work;
98                  u(m_action+1) = 1;
99
100                 // Evaluate the neural network
101                 Eigen::VectorXd y = transitionNetwork(u);
102
103                 // get the index of the max value from the output
104                 unsigned r, c;
105                 y.maxCoeff(&r, &c);
106                 m_action = r;
107
```

```
108                 // reset the clock
109                 m_timer.startTimer();
110             }
111         }
112 
113         // otherwise...
114         else {
115             // pull the trajectory waypoint
116             m_wp = m_trajectories.at(m_action).step(t);
117         }
118     }
119 }
120 
121 // ------------------------------------------------
122 // Get the knet force prediction
123 // ------------------------------------------------
124 void Knet::predictForce(Eigen::VectorXd x,
125                         Eigen::VectorXd v,
126                         Eigen::VectorXd& Mu,
127                         Eigen::MatrixXd& Sigma)
128 {
129    // specify the indices for preforming regression using the model
130     Eigen::VectorXd inputXV(4);
131     Eigen::VectorXi indexXV(4);
132     Eigen::VectorXi indexF(2);
133 
134     // specify the indices
135     indexXV << 0, 1, 2, 3;
136     indexF << 6, 7;
```

```
137
138     // cat the inputs to xv
139     inputXV.topRows(2) = x;
140     inputXV.bottomRows(2) = v;
141
142     // return values for the current action
143     m_actions.at(m_action).regression(inputXV, indexXV, indexF, Mu, Sigma);
144 }
145
146 // -------------------------------------------------
147 // Get the knet acceleration prediction
148 // -------------------------------------------------
149 void Knet::predictAccel(Eigen::VectorXd x,
150                         Eigen::VectorXd v,
151                         Eigen::VectorXd& Mu,
152                         Eigen::MatrixXd& Sigma)
153 {
154    // specify the indices for preforming regression using the model
155     Eigen::VectorXd inputXV(4);
156     Eigen::VectorXi indexXV(4);
157     Eigen::VectorXi indexA(2);
158
159     // specify the indices
160     indexXV << 0, 1, 2, 3;
161     indexA << 4, 5;
162
163     // cat the inputs to xv
164     inputXV.topRows(2) = x;
165     inputXV.bottomRows(2) = v;
```

```
166
167     // return values for the current action
168     m_actions.at(m_action).regression(inputXV, indexXV, indexA, Mu, Sigma);
169 }
170
171 // ------------------------------------------------
172 // Transition model (neural network)
173 // ------------------------------------------------
174 Eigen::VectorXd Knet::transitionNetwork(Eigen::VectorXd u) {
175
176     // Input layer
177     // Map Minimum and Maximum Input Processing Function
178     Eigen::VectorXd y1 = u - m_nnet.il_xoffset;
179     y1 = m_nnet.il_gain.cwiseProduct(y1);
180     y1 = y1 + m_nnet.il_ymin * Eigen::VectorXd::Ones(5);
181
182     // Hidden layer
183     // Sigmoid Symmetric Transfer Function
184     Eigen::VectorXd y2 = m_nnet.l1_W * y1 + m_nnet.l1_b;
185     y2 = 2 * (1 + (-2 * y2.array()).exp()).cwiseInverse() - 1;
186
187     // Output layer
188     // Competitive Soft Transfer Function (Softmax)
189     Eigen::VectorXd y = m_nnet.l2_W * y2 + m_nnet.l2_b;
190     double y0 = y.maxCoeff();
191     y = y - y0 * Eigen::VectorXd::Ones(4);
192     Eigen::VectorXd num = y.array().exp();
193     double den = num.array().sum();
194     if (den == 0) {
```

```
195         den = 1.0;
196     }
197     y = (1 / den) * num;
198     return y;
199 }
200 
201 // ------------------------------------------------
202 // Return the action index
203 // ------------------------------------------------
204 int Knet::getActionIndex() {
205     if (m_done) {
206         return -1;
207     } else {
208         return int(m_action);
209     }
210 }
```

Listing C.24: lib/Trajectory.h - Trajectory class header

```
1 /*
2  Copyright 2014-2019, Parker Owan
3  PhD Dissertation, University of Washington
4  */
5 
6 #ifndef TRAJECTORY_H_
7 #define TRAJECTORY_H_
8 
9 #include <cr/core>
10 #include <cr/control>
```

```
11
12 //-------------------------------------------------------------
13 // Trajectory Class
14 //-------------------------------------------------------------
15 class Trajectory
16 {
17     // constructor and destructor
18     public:
19         Trajectory() {};
20         ¬Trajectory() {
21             m_list.clear();
22         };
23
24     // waypoint exectution methods
25     public:
26
27         // check if it's done
28         bool isDone(double t);
29
30         // linear interpolation
31         cr::control::Waypoint step(double t);
32
33         // push back a waypoint
34         void add(cr::control::Waypoint wp) {
35             m_list.push_back(wp);
36         }
37
38     public:
39
```

```
40         // trajectory data
41         std::vector<cr::control::Waypoint> m_list;
42 };
43
44 #endif
```

Listing C.25: lib/Trajectory.cpp - Trajectory class source

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #include "Trajectory.h"
7
8  // ------------------------------------------------
9  // Trajectory interpolate
10 // ------------------------------------------------
11 cr::control::Waypoint Trajectory::step(double t)
12 {
13     // initialize a waypoint
14     cr::control::Waypoint wp;
15
16     // get the number of waypoints in the list
17     unsigned n = m_list.size();
18
19     // if there is more than one waypoint in the list...
20     if (n > 1) {
21
```

```
22          // check if we haven't started...
23          if (t ≤0) {
24
25              // assign the first waypoint
26              wp = m_list.at(0);
27          }
28          // check if we've ended...
29          else if (t ≥m_list.at(n-1).time) {
30
31              // assign the last waypoint
32              wp = m_list.at(n-1);
33              wp.time = t;
34          }
35          // otherwise, let's interpolate
36          else {
37
38              // define the two waypoints
39              cr::control::Waypoint wp0, wp1;
40
41              // set up the starting indices to check
42              unsigned idx0 = 0;
43              unsigned idx1 = 1;
44
45              // find which indices we're in between
46              bool done = false;
47              while (!done) {
48                  wp0 = m_list.at(idx0);
49                  wp1 = m_list.at(idx1);
50                  double t0 = wp0.time;
```

```
51                double t1 = wp1.time;
52                if ((t ≥t0) && (t < t1)) {
53                    done = true;
54                } else {
55                    idx0++;
56                    idx1++;
57                }
58            }
59
60            // interpolate in between the two waypoints
61            double         t0 = wp0.time;
62            double         t1 = wp1.time;
63            double         dt = t1 - t0;
64            Eigen::VectorXd x0 = wp0.position;
65            Eigen::VectorXd x1 = wp1.position;
66            Eigen::VectorXd v0 = wp0.velocity;
67            Eigen::VectorXd v1 = wp1.velocity;
68
69            wp.time = t;
70            wp.position = (t - t0) * (x1 - x0) / dt + x0;
71            wp.velocity = (t - t0) * (v1 - v0) / dt + v0;
72        }
73    }
74    // if there is only one waypoint in the list...
75    else if (m_list.size() == 1) {
76        wp = m_list.at(0);
77    }
78    return wp;
79 }
```

```
80
81 // ------------------------------------------------
82 // Trajectory interpolate
83 // ------------------------------------------------
84 bool Trajectory::isDone(double t)
85 {
86     unsigned n = m_list.size();
87     if (n > 0) {
88         if (t ≥m_list.at(n-1).time) {
89             return true;
90         } else {
91             return false;
92         }
93     } else {
94         return true;
95     }
96 }
```

Listing C.26: lib/DynamicsStep.h - Robot dynamics simulation class

```
1 /*
2  Copyright 2014-2019, Parker Owan
3  PhD Dissertation, University of Washington
4  */
5
6 #ifndef DYNAMICS_STEP_H_
7 #define DYNAMICS_STEP_H_
8
9 #include <cr/core>
```

```
10 #include <cr/math>
11
12 // ------------------------------------------------
13 // Dynamics simulation step
14 // ------------------------------------------------
15 class DynamicsStep : public cr::core::Step
16 {
17
18     // constructor and destructor
19     public:
20         DynamicsStep(double dt) { m_dt = dt; }
21         ¬DynamicsStep() {};
22
23     // derived step elements
24     public:
25         virtual void step();
26
27     // subroutines
28     public:
29         void setCommand(Eigen::VectorXd u) { m_u = u; }
30         void setPosition(Eigen::VectorXd q) { m_q = q; m_qm = q; }
31         void setVelocity(Eigen::VectorXd qd) { m_qd = qd; m_qmd = qd; }
32         Eigen::VectorXd getPosition() { return m_q; }
33         Eigen::VectorXd getVelocity() { return m_qd; }
34         Eigen::VectorXd getMotorPosition() { return m_qm; }
35         Eigen::VectorXd getMotorVelocity() { return m_qmd; }
36
37     // properties
38     private:
```

```
39
40          //! clock
41          cr::core::Clock m_timer;
42
43          //! sample rate
44          double m_dt;
45
46          //! Reference states
47          Eigen::VectorXd m_u;
48          Eigen::VectorXd m_q;
49          Eigen::VectorXd m_qd;
50          Eigen::VectorXd m_qm;
51          Eigen::VectorXd m_qmd;
52  };
53
54  #endif
```

Listing C.27: lib/DynamicsStep.cpp - Robot dynamics simulation class

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #include "DynamicsStep.h"
7
8  // Declare a dynamics function
9  Eigen::VectorXd robotDyamics(double t,
10                               Eigen::VectorXd x,
```

```
11                              Eigen::VectorXd u)
12 {
13     // Pull states from augmented state
14     Eigen::VectorXd qm = x.segment(0,3);    // motor position
15     Eigen::VectorXd qmd = x.segment(3,3);   // motor velocity
16     Eigen::VectorXd q  = x.segment(6,3);    // output position
17     Eigen::VectorXd qd = x.segment(9,3);    // output velocity
18     Eigen::VectorXd xdot = x;
19
20     //! Motor parameters
21     double motrJ = 4.993e-8; // original
22     double motrB = 3.031e-7; // original
23     double motrN = 765.75;    // Gear Ratio
24     double K = 5000.0;       // Stiffness N / rad
25
26     //! Link parameters (these are rough and do not predict the behavior well)
27     double m1 = 0.68;    // (kg) mass link 1
28     double m2 = 0.68;    // (kg) mass link 2
29     double m3 = 0.375;   // (kg) mass link 3
30     double I1 = 1870e-6; // (kg-m^2) inertia link 1
31     double I2 = 1870e-6; // (kg-m^2) inertia link 2
32     double I3 = 1200e-6; // (kg-m^2) inertia link 3
33     double r1 = 0.136;   // (m) com link 1
34     double r2 = 0.136;   // (m) com link 2
35     double r3 = 0.014;   // (m) com link 3
36     double l1 = 0.16905; // (m) link 1
37     double l2 = 0.18008; // (m) link 2
38     double l3 = 0.15520; // (m) tool
39
```

```
40    // MOTOR DYNAMICS
41    double Jm = motrJ * pow(motrN, 2);
42    double Bm = motrB * pow(motrN, 2);
43
44    // THIS IS A MORE COMPLEX, SERIES ELASTIC COUPLED MODEL
45    double a = I1 + I2 + I3 + pow(l1,2)*(m2 + m3) + m1*pow(r1,2) +
46        m2*pow(r2,2) + m3*(pow(l2,2) + pow(r3,2));
47    double b = l1*(l2*m3 + m2*r2);
48    double d = l2*m3*r3;
49    double e = l1*m3*r3;
50    double f = I2 + I3 + m2*pow(r2,2) + m3*(pow(l2,2) + pow(r3,2));
51    double g = I3 + m3*pow(r3,2);
52
53    // Mass matrix (robot only)
54    Eigen::MatrixXd M(3,3);
55    M << a + 2*(b*cos(q(1)) + d*cos(q(2)) + e*cos(q(1)+q(2))),
56         f + b*cos(q(1)) + 2*d*cos(q(2)) + e*cos(q(1)+q(2)),
57         g + d*cos(q(2)) + e*cos(q(1)+q(2)),
58         f + b*cos(q(1)) + 2*d*cos(q(2)) + e*cos(q(1)+q(2)),
59         g + f + 2*d*cos(q(2)),
60         g + d*cos(q(2)),
61         g + d*cos(q(2)) + e*cos(q(1)+q(2)),
62         g + d*cos(q(2)),
63         g;
64
65    // Mass matrix (robot + motor inertia)
66    Eigen::MatrixXd M2(3,3);
67    M2 << Jm + a + 2*(b*cos(q(1)) + d*cos(q(2)) + e*cos(q(1)+q(2))),
68         f + b*cos(q(1)) + 2*d*cos(q(2)) + e*cos(q(1)+q(2)),
```

```
69          g + d*cos(q(2)) + e*cos(q(1)+q(2)),
70          f + b*cos(q(1)) + 2*d*cos(q(2)) + e*cos(q(1)+q(2)),
71          Jm + g + f + 2*d*cos(q(2)),
72          g + d*cos(q(2)),
73          g + d*cos(q(2)) + e*cos(q(1)+q(2)),
74          g + d*cos(q(2)),
75          Jm + g;
76
77      // Coriolis forces (3 x 1)
78      Eigen::VectorXd C(3);
79      C << -(b*sin(q(1)) + e*sin(q(1)+q(2)))*qd(1)*(2*qd(0)+qd(1))
80                - 2*(d*sin(q(2)) + e*sin(q(1)+q(2)))*(qd(0)+qd(1))*qd(2)
81                - (d*sin(q(2)) + e*sin(q(1)+q(2)))*pow(qd(2),2),
82             (b*sin(q(1)) + e*sin(q(1)+q(2)))*pow(qd(0),2)
83                - 2*d*sin(q(2))*(qd(0)+qd(1))*qd(2)
84                - d*sin(q(2))*pow(qd(2),2),
85             e*cos(q(2))*sin(q(1))*pow(qd(0),2)
86                + e*cos(q(1))*sin(q(2))*pow(qd(0),2)
87                + d*sin(q(2))*pow(qd(0) + qd(1),2);
88
89      // Compute the output state
90      // xdot.segment(0, 3) = qmd; // motor velocity
91      // motor dynamics
92      // xdot.segment(3, 3) = - (K/Jm) * (qm - q) -(Bm/Jm) * qmd + (1/Jm) * u;
93      // xdot.segment(6, 3) = qd; // output velocity
94      // xdot.segment(9, 3) = M.inverse() * (- (K) * (q - qm)); // output ...
            dynamics
95
96      // Reduced order - robot dynamics
```

```
97      // xdot.segment(6, 3) = qd; // output velocity
98      // xdot.segment(9, 3) = M2.inverse() * (- Bm * qd + u); // output dynamics
99
100     // Reduced order - motor dynamics
101     xdot.segment(6, 3) = qd; // output velocity
102     xdot.segment(9, 3) = - (Bm / Jm) * qd + (1 / Jm) * u; // motor dynamics
103
104     // return the state derivative
105     return xdot;
106 }
107
108 // ------------------------------------------------
109 // Step
110 // ------------------------------------------------
111 void DynamicsStep::step()
112 {
113     // construct the state vector
114     Eigen::VectorXd x;
115     x.setZero(12);
116     x.segment(0, 3) = m_qm;  // motor position
117     x.segment(3, 3) = m_qmd; // motor velocity
118     x.segment(6, 3) = m_q;   // output position
119     x.segment(9, 3) = m_qd;  // output velocity
120
121     // Step the RK4 solver
122     x = cr::math::Integration::rungeKuttaStep(*robotDyamics, 0, x, m_u, m_dt);
123
124     // Deconstruct the state vector
125     m_qm  = x.segment(0,3);  // motor position
```

```
126     m_qmd  = x.segment(3,3);  // motor velocity
127     m_q    = x.segment(6,3);  // output position
128     m_qd   = x.segment(9,3);  // output velocity
129 }
```

Listing C.28: lib/AtiFeedback.h - ATI force sensor class header

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #ifndef ATI_FEEDBACK_H_
7  #define ATI_FEEDBACK_H_
8
9  #include <cr/core>
10
11 // NOTE: We need to define this so that we only use error code headers
12 // There are compiler flags to NOT make use of the compiled boost ...
       dependencies,
13 // i.e. -DBOOST_DATE_TIME_NO_LIB -DBOOST_REGEX_NO_LIB -DBOOST_SYSTEM_NO_LIB
14 #define BOOST_DATE_TIME_NO_LIB
15 #define BOOST_REGEX_NO_LIB
16 #define BOOST_SYSTEM_NO_LIB
17
18 #define BOOST_ERROR_CODE_HEADER_ONLY
19 #include <boost/system/error_code.hpp>
20
21 #include "boost/asio.hpp"
```

```
22 #include <boost/array.hpp>
23 #include <boost/bind.hpp>
24 #include <iostream>
25
26 // using the boost udp protocol
27 using boost::asio::ip::udp;
28
29 //------------------------------------------------------------
30 // ATI FEEDBACK WRAPPER
31 //------------------------------------------------------------
32 class AtiFeedback : public cr::core::Step
33 {
34
35     // constructor and destructor
36     public:
37         AtiFeedback(boost::asio::io_context& ioContext);
38         ¬AtiFeedback();
39
40     // controls
41     public:
42         void connect(std::string address);
43         void applyBias();
44
45     // derived step elements
46     public:
47         void step();
48
49     // Data access elements
50     public:
```

```
51        Eigen::VectorXd getFbkEffort() {
52            Eigen::VectorXd f(6);
53            for (unsigned i = 0; i < 6; i++) {
54                f(i) = m_feedback[i];
55            }
56            return f;
57        }
58
59    // properties
60    private:
61        bool m_connected = false;
62        double m_feedback[6] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
63        double m_scaleFactor[6] = {
64            1.0 / 224808.9375,
65            1.0 / 224808.9375,
66            1.0 / 224808.9375,
67            1.0 / 8850746.0,
68            1.0 / 8850746.0,
69            1.0 / 8850746.0 };
70
71        // boost functions
72        // boost::asio::io_context m_ioContext;
73        udp::socket m_socket;
74        udp::resolver m_resolver;
75        udp::resolver::results_type m_endpoints;
76        std::string m_address = "192.168.1.1";
77
78    // Fix the static size matrices to be 128-bit aligned
79    // ...
```

```
        https://eigen.tuxfamily.org/dox/group__TopicStructHavingEigenMembers.html
80      // public:
81          // EIGEN_MAKE_ALIGNED_OPERATOR_NEW
82  };
83
84  #endif
```

Listing C.29: lib/AtiFeedback.cpp - ATI force sensor class source

```
1  /*
2   Copyright 2014-2019, Parker Owan
3   PhD Dissertation, University of Washington
4   */
5
6  #include "AtiFeedback.h"
7
8  // ATI specific definitions
9  #define CMD_STREAM 2 // Command code 2 starts streaming
10 #define CMD_BIAS 42 // Command code 42 sets the software bias
11 #define NUM_SAMPLES 1 // Will send 1 sample before stopping
12
13 #define SEND_MSG_LENGTH 8
14 #define RECV_MSG_LENGTH 36
15
16 /* Typedefs used so integer sizes are more explicit */
17 typedef unsigned int uint32;
18 typedef int int32;
19 typedef unsigned short uint16;
20 typedef short int16;
```

```
21
22 //-------------------------------------------------------------
23 // ATI FEEDBACK WRAPPER
24 //-------------------------------------------------------------
25
26 //! constructor
27 AtiFeedback::AtiFeedback(boost::asio::io_context& ioContext)
28     : m_socket(ioContext, udp::endpoint(udp::v4(), 0)), m_resolver(ioContext)
29 { }
30
31 //! destructor
32 AtiFeedback::¬AtiFeedback() {}
33
34 //! send the command to bias the force/torque sensor offset
35 void AtiFeedback::connect(std::string address)
36 {
37     // push the address to m_address
38     m_address = address;
39
40     // create the endpoint object
41     m_endpoints = m_resolver.resolve(udp::v4(), m_address, "49152");
42
43     // set connected flag high
44     m_connected = true;
45 }
46
47 //! send the command to bias the force/torque sensor offset
48 void AtiFeedback::applyBias()
49 {
```

```
50      if (m_connected)
51      {
52          // ATI - per table 9.1 in Net F/T user manual.
53          char request[SEND_MSG_LENGTH];
54          *(uint16*)&request[0] = htons(0x1234);
55          *(uint16*)&request[2] = htons(CMD_BIAS);
56          *(uint32*)&request[4] = htonl(NUM_SAMPLES);
57
58          // send the message to the socket
59          m_socket.send_to(boost::asio::buffer(request, SEND_MSG_LENGTH),
60              *m_endpoints.begin());
61      }
62  }
63
64  //! step
65  void AtiFeedback::step()
66  {
67      if (m_connected)
68      {
69          // ATI - per table 9.1 in Net F/T user manual.
70          char request[SEND_MSG_LENGTH];
71          *(uint16*)&request[0] = htons(0x1234);
72          *(uint16*)&request[2] = htons(CMD_STREAM);
73          *(uint32*)&request[4] = htonl(NUM_SAMPLES);
74
75          // send the message to the socket
76          m_socket.send_to(boost::asio::buffer(request, SEND_MSG_LENGTH),
77              *m_endpoints.begin());
78
```

```
79          // setup the receive buffer & endpoint
80          char reply[RECV_MSG_LENGTH];
81          udp::endpoint sender_endpoint;
82
83          // receive the message
84          size_t reply_length = m_socket.receive_from(
85              boost::asio::buffer(reply, RECV_MSG_LENGTH), sender_endpoint);
86
87          // parse the message
88          uint32 rdt_sequence = ntohl(*(uint32*)&reply[0]);
89          uint32 ft_sequence = ntohl(*(uint32*)&reply[4]);
90          uint32 status = ntohl(*(uint32*)&reply[8]);
91          int32 FTData[6];
92          for (int i = 0; i < 6; i++) {
93              FTData[i] = ntohl(*(int32*)&reply[12 + i * 4]);
94          }
95
96          // convert the data to the feedback vector
97          for (int i = 0; i < 6; i++) {
98              m_feedback[i] = m_scaleFactor[i] * double(FTData[i]);
99          }
100     }
101 }
```

C.2.4 Cleaning policy constants

Listing C.30: include/KnetConstants.h - Cleaning policy parameter header

```
1 // Parameter file for Knet.h
```

```
2  // Autogenerated by knet.m (method: writeParameterFile)
3  // Copyright (2018) Parker Owan
4
5  #ifndef KNET_CONSTANTS_H_
6  #define KNET_CONSTANTS_H_
7
8  #include <cr/noise>
9  #include <cr/control>
10 #include "Trajectory.h"
11 #include "Knet.h"
12
13 // DIMENSIONS
14 #define KNET_DIM         2
15 #define KNET_ACTIONS     4
16 #define KNET_MIXTURES    8
17 #define KNET_FIRST_ACTION  2
18 #define KNET_LAST_ACTION  3
19
20 // TRAJECTORIES
21 void loadKnetTrajectories(Trajectory (&traj)[KNET_ACTIONS]) {
22
23     cr::control::Waypoint wp;
24     Eigen::VectorXd x(KNET_DIM);
25     Eigen::VectorXd v(KNET_DIM);
26
27     // Trajectory 0, Point 0
28     x << +0.0010637064, -0.0009677516;
29     v << -0.2663741456, -0.0136744051;
30     wp.time = +0.0000000000;
```

```
31      wp.position = x;
32      wp.velocity = v;
33      traj[0].add(wp);
34
35      // Trajectory 0, Point 1
36      x << -0.0040984019, -0.0017752659;
37      v << -0.2421912559, -0.0110010953;
38      wp.time = +0.0195766441;
39      wp.position = x;
40      wp.velocity = v;
41      traj[0].add(wp);
42
43      // Trajectory 0, Point 2
44      x << -0.0083991747, -0.0024030649;
45      v << -0.1936641328, -0.0091611490;
46      wp.time = +0.0391532883;
47      wp.position = x;
48      wp.velocity = v;
49      traj[0].add(wp);
50
51      // Trajectory 0, Point 3
52      x << -0.0115617892, -0.0027355400;
53      v << -0.1184914032, -0.0034811519;
54      wp.time = +0.0587299324;
55      wp.position = x;
56      wp.velocity = v;
57      traj[0].add(wp);
58
59      // Trajectory 0, Point 4
```

```
60      x << -0.0132783466, -0.0027996773;
61      v << -0.0396251537, +0.0022365478;
62      wp.time = +0.0783065765;
63      wp.position = x;
64      wp.velocity = v;
65      traj[0].add(wp);
66
67      // Trajectory 0, Point 5
68      x << -0.0135046439, -0.0026658882;
69      v << +0.0405540992, +0.0068124437;
70      wp.time = +0.0978832206;
71      wp.position = x;
72      wp.velocity = v;
73      traj[0].add(wp);
74
75      // Trajectory 0, Point 6
76      x << -0.0122620550, -0.0023464950;
77      v << +0.1210667688, +0.0104299923;
78      wp.time = +0.1174598648;
79      wp.position = x;
80      wp.velocity = v;
81      traj[0].add(wp);
82
83      // Trajectory 0, Point 7
84      x << -0.0095944783, -0.0018348988;
85      v << +0.2005518964, +0.0134832341;
86      wp.time = +0.1370365089;
87      wp.position = x;
88      wp.velocity = v;
```

```
89      traj[0].add(wp);
90
91      // Trajectory 0, Point 8
92      x << -0.0056832544, -0.0012196955;
93      v << +0.2561414321, +0.0191356356;
94      wp.time = +0.1566131530;
95      wp.position = x;
96      wp.velocity = v;
97      traj[0].add(wp);
98
99      // Trajectory 0, Point 9
100     x << -0.0012817158, -0.0003249576;
101     v << +0.2796660287, +0.0285286017;
102     wp.time = +0.1761897971;
103     wp.position = x;
104     wp.velocity = v;
105     traj[0].add(wp);
106
107     // Trajectory 0, Point 10
108     x << +0.0033513420, +0.0009654348;
109     v << +0.2919516322, +0.0371820206;
110     wp.time = +0.1957664413;
111     wp.position = x;
112     wp.velocity = v;
113     traj[0].add(wp);
114
115     // Trajectory 1, Point 0
116     x << +0.0033513925, +0.0009654056;
117     v << +0.2919516483, +0.0371819543;
```

```
118     wp.time = +0.0000000000;
119     wp.position = x;
120     wp.velocity = v;
121     traj[1].add(wp);
122
123     // Trajectory 1, Point 1
124     x << +0.0095378980, +0.0020680793;
125     v << +0.2958248262, +0.0432307068;
126     wp.time = +0.0202467300;
127     wp.position = x;
128     wp.velocity = v;
129     traj[1].add(wp);
130
131     // Trajectory 1, Point 2
132     x << +0.0156383082, +0.0031594719;
133     v << +0.2872400762, +0.0450377732;
134     wp.time = +0.0404934600;
135     wp.position = x;
136     wp.velocity = v;
137     traj[1].add(wp);
138
139     // Trajectory 1, Point 3
140     x << +0.0211508373, +0.0040754182;
141     v << +0.2441670873, +0.0316992659;
142     wp.time = +0.0607401900;
143     wp.position = x;
144     wp.velocity = v;
145     traj[1].add(wp);
146
```

```
147     // Trajectory 1, Point 4
148     x << +0.0254087006, +0.0045188569;
149     v << +0.1733668988, +0.0053029039;
150     wp.time = +0.0809869200;
151     wp.position = x;
152     wp.velocity = v;
153     traj[1].add(wp);
154
155     // Trajectory 1, Point 5
156     x << +0.0280731705, +0.0043463180;
157     v << +0.0920945397, -0.0259563825;
158     wp.time = +0.1012336500;
159     wp.position = x;
160     wp.velocity = v;
161     traj[1].add(wp);
162
163     // Trajectory 1, Point 6
164     x << +0.0290982552, +0.0034924888;
165     v << +0.0183791158, -0.0546310605;
166     wp.time = +0.1214803799;
167     wp.position = x;
168     wp.velocity = v;
169     traj[1].add(wp);
170
171     // Trajectory 1, Point 7
172     x << +0.0289662649, +0.0023027674;
173     v << -0.0295006487, -0.0559770183;
174     wp.time = +0.1417271099;
175     wp.position = x;
```

```
176     wp.velocity = v;
177     traj[1].add(wp);
178
179     // Trajectory 1, Point 8
180     x << +0.0279104188, +0.0012943300;
181     v << -0.0779720285, -0.0332827481;
182     wp.time = +0.1619738399;
183     wp.position = x;
184     wp.velocity = v;
185     traj[1].add(wp);
186
187     // Trajectory 1, Point 9
188     x << +0.0256172711, +0.0007862272;
189     v << -0.1495989778, -0.0117361782;
190     wp.time = +0.1822205699;
191     wp.position = x;
192     wp.velocity = v;
193     traj[1].add(wp);
194
195     // Trajectory 1, Point 10
196     x << +0.0219355122, +0.0005121405;
197     v << -0.2101368958, -0.0082951435;
198     wp.time = +0.2024672999;
199     wp.position = x;
200     wp.velocity = v;
201     traj[1].add(wp);
202
203     // Trajectory 1, Point 11
204     x << +0.0172045362, +0.0002058924;
```

```
    v << -0.2531586877, -0.0145459742;
    wp.time = +0.2227140299;
    wp.position = x;
    wp.velocity = v;
    traj[1].add(wp);

    // Trajectory 1, Point 12
    x << +0.0119221728, -0.0002022320;
    v << -0.2716636076, -0.0177413313;
    wp.time = +0.2429607599;
    wp.position = x;
    wp.velocity = v;
    traj[1].add(wp);

    // Trajectory 1, Point 13
    x << +0.0064663821, -0.0006150455;
    v << -0.2751676631, -0.0162499079;
    wp.time = +0.2632074899;
    wp.position = x;
    wp.velocity = v;
    traj[1].add(wp);

    // Trajectory 1, Point 14
    x << +0.0010638064, -0.0009677766;
    v << -0.2663759678, -0.0136745625;
    wp.time = +0.2834542199;
    wp.position = x;
    wp.velocity = v;
    traj[1].add(wp);
```

```
234
235     // Trajectory 2, Point 0
236     x << -0.0364319950, -0.0016036763;
237     v << +0.0003083310, +0.0005350945;
238     wp.time = +0.0000000000;
239     wp.position = x;
240     wp.velocity = v;
241     traj[2].add(wp);
242
243     // Trajectory 2, Point 1
244     x << -0.0357935463, -0.0015646799;
245     v << -0.0005792611, +0.0023318431;
246     wp.time = +0.0206141484;
247     wp.position = x;
248     wp.velocity = v;
249     traj[2].add(wp);
250
251     // Trajectory 2, Point 2
252     x << -0.0351648999, -0.0014801624;
253     v << -0.0005891452, +0.0042546459;
254     wp.time = +0.0412282969;
255     wp.position = x;
256     wp.velocity = v;
257     traj[2].add(wp);
258
259     // Trajectory 2, Point 3
260     x << -0.0345291188, -0.0013526551;
261     v << +0.0004396737, +0.0066865726;
262     wp.time = +0.0618424453;
```

```
263     wp.position = x;
264     wp.velocity = v;
265     traj[2].add(wp);
266
267     // Trajectory 2, Point 4
268     x << -0.0338225977, -0.0011677034;
269     v << +0.0039242041, +0.0103685406;
270     wp.time = +0.0824565937;
271     wp.position = x;
272     wp.velocity = v;
273     traj[2].add(wp);
274
275     // Trajectory 2, Point 5
276     x << -0.0326040545, -0.0008100926;
277     v << +0.0302058296, +0.0141844784;
278     wp.time = +0.1030707421;
279     wp.position = x;
280     wp.velocity = v;
281     traj[2].add(wp);
282
283     // Trajectory 2, Point 6
284     x << -0.0305866753, -0.0003258730;
285     v << +0.0629920821, +0.0175266022;
286     wp.time = +0.1236848906;
287     wp.position = x;
288     wp.velocity = v;
289     traj[2].add(wp);
290
291     // Trajectory 2, Point 7
```

```
292     x << -0.0277035970, +0.0002854761;
293     v << +0.0972263066, +0.0205779213;
294     wp.time = +0.1442990390;
295     wp.position = x;
296     wp.velocity = v;
297     traj[2].add(wp);
298
299     // Trajectory 2, Point 8
300     x << -0.0238239065, +0.0009940112;
301     v << +0.1341536690, +0.0229933388;
302     wp.time = +0.1649131874;
303     wp.position = x;
304     wp.velocity = v;
305     traj[2].add(wp);
306
307     // Trajectory 2, Point 9
308     x << -0.0179879145, +0.0016599935;
309     v << +0.1913733293, +0.0239930795;
310     wp.time = +0.1855273359;
311     wp.position = x;
312     wp.velocity = v;
313     traj[2].add(wp);
314
315     // Trajectory 2, Point 10
316     x << -0.0108210936, +0.0019912607;
317     v << +0.2520672375, +0.0263366288;
318     wp.time = +0.2061414843;
319     wp.position = x;
320     wp.velocity = v;
```

```
321     traj[2].add(wp);
322
323     // Trajectory 2, Point 11
324     x << -0.0035816617, +0.0017196702;
325     v << +0.2793168921, +0.0313060625;
326     wp.time = +0.2267556327;
327     wp.position = x;
328     wp.velocity = v;
329     traj[2].add(wp);
330
331     // Trajectory 2, Point 12
332     x << +0.0033516845, +0.0009654579;
333     v << +0.2919541912, +0.0371831164;
334     wp.time = +0.2473697811;
335     wp.position = x;
336     wp.velocity = v;
337     traj[2].add(wp);
338
339     // Trajectory 3, Point 0
340     x << +0.0010638032, -0.0009677427;
341     v << -0.2663756037, -0.0136744952;
342     wp.time = +0.0000000000;
343     wp.position = x;
344     wp.velocity = v;
345     traj[3].add(wp);
346
347     // Trajectory 3, Point 1
348     x << -0.0029659630, -0.0014214367;
349     v << -0.2552959911, -0.0112298677;
```

```
    wp.time = +0.0116138818;
    wp.position = x;
    wp.velocity = v;
    traj[3].add(wp);

    // Trajectory 3, Point 2
    x << -0.0077168192, -0.0017729838;
    v << -0.2359575258, -0.0090637528;
    wp.time = +0.0232277636;
    wp.position = x;
    wp.velocity = v;
    traj[3].add(wp);

    // Trajectory 3, Point 3
    x << -0.0125761632, -0.0020122997;
    v << -0.2046246964, -0.0070703447;
    wp.time = +0.0348416454;
    wp.position = x;
    wp.velocity = v;
    traj[3].add(wp);

    // Trajectory 3, Point 4
    x << -0.0171190261, -0.0020530880;
    v << -0.1599680964, -0.0038883911;
    wp.time = +0.0464555272;
    wp.position = x;
    wp.velocity = v;
    traj[3].add(wp);

```

```
379     // Trajectory 3, Point 5
380     x << -0.0209885427, -0.0020702336;
381     v << -0.1152170379, -0.0008869289;
382     wp.time = +0.0580694090;
383     wp.position = x;
384     wp.velocity = v;
385     traj[3].add(wp);
386
387     // Trajectory 3, Point 6
388     x << -0.0241335091, -0.0019774233;
389     v << -0.0823934145, +0.0003347217;
390     wp.time = +0.0696832908;
391     wp.position = x;
392     wp.velocity = v;
393     traj[3].add(wp);
394
395     // Trajectory 3, Point 7
396     x << -0.0266422336, -0.0019209289;
397     v << -0.0617143401, +0.0008513562;
398     wp.time = +0.0812971726;
399     wp.position = x;
400     wp.velocity = v;
401     traj[3].add(wp);
402
403     // Trajectory 3, Point 8
404     x << -0.0287636433, -0.0018255220;
405     v << -0.0449394728, +0.0013785880;
406     wp.time = +0.0929110544;
407     wp.position = x;
```

```
408     wp.velocity = v;
409     traj[3].add(wp);
410
411     // Trajectory 3, Point 9
412     x << -0.0306236625, -0.0017087371;
413     v << -0.0289329451, +0.0022391332;
414     wp.time = +0.1045249361;
415     wp.position = x;
416     wp.velocity = v;
417     traj[3].add(wp);
418
419     // Trajectory 3, Point 10
420     x << -0.0321631642, -0.0015543151;
421     v << -0.0137558494, +0.0032119331;
422     wp.time = +0.1161388179;
423     wp.position = x;
424     wp.velocity = v;
425     traj[3].add(wp);
426
427     // Trajectory 3, Point 11
428     x << -0.0364319871, -0.0016036778;
429     v << +0.0003083273, +0.0005350931;
430     wp.time = +0.1277526997;
431     wp.position = x;
432     wp.velocity = v;
433     traj[3].add(wp);
434
435 }
436
```

```
437 // MIXTURE MODELS
438 void loadMixtureModels(cr::noise::Gmm (&gmm)[KNET_ACTIONS]) {
439
440     cr::noise::NoiseGaussian gaussian[KNET_MIXTURES];
441     Eigen::VectorXd Mu(4 * KNET_DIM);
442     Eigen::MatrixXd Sigma(4 * KNET_DIM, 4 * KNET_DIM);
443
444     // Gaussian Cluster 0
445     Mu << +0.0033512931, +0.0009654387, +0.2919515604, +0.0371819925, ...
            +0.4199890302, +0.3532990938, -10.2508864606, +0.1074844808;
446     Sigma << +0.0001196196, +0.0000041460, +0.0001469398, +0.0001181301, ...
            -0.0082368961, -0.0009264982, -0.0130584540, -0.0033063695,
447 +0.0000041460, +0.0000145475, -0.0000024254, -0.0000017790, -0.0012151828, ...
       -0.0011828013, -0.0010403628, -0.0020792425,
448 +0.0001469398, -0.0000024254, +0.0009536830, +0.0002887709, -0.0094141440, ...
       +0.0025216407, -0.0294139369, +0.0006844835,
449 +0.0001181301, -0.0000017790, +0.0002887709, +0.0006654546, -0.0091262174, ...
       -0.0012741927, -0.0104366003, -0.0038654486,
450 -0.0082368961, -0.0012151828, -0.0094141440, -0.0091262174, +1.2954494104, ...
       +0.4023037768, +1.3185428385, +0.8648585165,
451 -0.0009264982, -0.0011828013, +0.0025216407, -0.0012741927, +0.4023037768, ...
       +0.5721925372, +0.2742826176, +0.7268658846,
452 -0.0130584540, -0.0010403628, -0.0294139369, -0.0104366003, +1.3185428385, ...
       +0.2742826176, +2.6505277879, +0.7787446304,
453 -0.0033063695, -0.0020792425, +0.0006844835, -0.0038654486, +0.8648585165, ...
       +0.7268658846, +0.7787446304, +1.3113029572;
454     gaussian[0].setParameters(Sigma, Mu);
455
456     // Gaussian Cluster 1
```

```
457     Mu << -0.0321317223, -0.0004782226, -0.0215617752, +0.0099417308, ...
            +1.4921159417, +0.0979351988, +0.0808482765, -0.1108622395;
458     Sigma << +0.0000382607, +0.0000009528, -0.0000488668, -0.0000022495, ...
            +0.0001165222, -0.0004945699, +0.0018254024, -0.0006041134,
459 +0.0000009528, +0.0000119538, +0.0000372728, +0.0000069522, +0.0000435284, ...
        -0.0000546095, -0.0002518748, -0.0000189845,
460 -0.0000488668, +0.0000372728, +0.0126905197, +0.0011436090, +0.0198677786, ...
        +0.0077385506, -0.0612136209, +0.0129729026,
461 -0.0000022495, +0.0000069522, +0.0011436090, +0.0002317388, +0.0044008817, ...
        +0.0014402192, -0.0062896486, +0.0008385352,
462 +0.0001165222, +0.0000435284, +0.0198677786, +0.0044008817, +1.4942283110, ...
        +0.0310046861, -0.2222758905, +0.0768739824,
463 -0.0004945699, -0.0000546095, +0.0077385506, +0.0014402192, +0.0310046861, ...
        +0.1154533567, -0.0446548699, +0.0273323123,
464 +0.0018254024, -0.0002518748, -0.0612136209, -0.0062896486, -0.2222758905, ...
        -0.0446548699, +1.0831401490, -0.1151907487,
465 -0.0006041134, -0.0000189845, +0.0129729026, +0.0008385352, +0.0768739824, ...
        +0.0273323123, -0.1151907487, +0.0478382197;
466     gaussian[1].setParameters(Sigma, Mu);
467
468     // Gaussian Cluster 2
469     Mu << +0.0010637268, -0.0009677523, -0.2663742312, -0.0136744374, ...
            +0.7863969865, +0.1264283167, +10.0477204982, -1.9950374401;
470     Sigma << +0.0001038499, +0.0000012066, -0.0000821265, -0.0000794974, ...
            -0.0107056017, -0.0003100483, +0.0188741817, -0.0060095462,
471 +0.0000012066, +0.0000127087, -0.0000027355, -0.0000017040, -0.0003400605, ...
        -0.0002223605, +0.0003508769, -0.0008649130,
472 -0.0000821265, -0.0000027355, +0.0012153204, +0.0000813077, +0.0318306916, ...
        -0.0032935228, -0.0238310823, +0.0108717545,
```

```
473 -0.0000794974, -0.0000017040, +0.0000813077, +0.0002145787, +0.0074569477, ...
      +0.0002317973, -0.0172429210, +0.0039728747,
474 -0.0107056017, -0.0003400605, +0.0318306916, +0.0074569477, +2.0127877313, ...
      -0.0823738498, -2.2271389976, +0.8863489483,
475 -0.0003100483, -0.0002223605, -0.0032935228, +0.0002317973, -0.0823738498, ...
      +0.1769215308, -0.0286877060, +0.1043244299,
476 +0.0188741817, +0.0003508769, -0.0238310823, -0.0172429210, -2.2271389976, ...
      -0.0286877060, +4.3677858567, -1.2602770240,
477 -0.0060095462, -0.0008649130, +0.0108717545, +0.0039728747, +0.8863489483, ...
      +0.1043244299, -1.2602770240, +0.7227423709;
478     gaussian[2].setParameters(Sigma, Mu);
479
480     // Gaussian Cluster 3
481     Mu << +0.0256298479, +0.0032232170, +0.1779760923, +0.0094164631, ...
           -3.7320644668, -1.4054830040, -16.2112641642, -2.8308384496;
482     Sigma << +0.0000277840, +0.0000005281, -0.0002041385, -0.0000843818, ...
           -0.0027409947, -0.0014635590, -0.0105593596, -0.0023416459,
483 +0.0000005281, +0.0000124118, -0.0000078208, -0.0000072891, -0.0003975424, ...
      -0.0003965298, -0.0013988296, -0.0009640527,
484 -0.0002041385, -0.0000078208, +0.0056557481, +0.0019033867, +0.0322877446, ...
      +0.0151499904, +0.0203939520, +0.0380591700,
485 -0.0000843818, -0.0000072891, +0.0019033867, +0.0010830809, +0.0145572976, ...
      +0.0054859459, +0.0327833831, +0.0126164179,
486 -0.0027409947, -0.0003975424, +0.0322877446, +0.0145572976, +0.8299497990, ...
      +0.4000830066, +2.0266811084, +0.6753549493,
487 -0.0014635590, -0.0003965298, +0.0151499904, +0.0054859459, +0.4000830066, ...
      +0.4395810926, +1.0930287679, +0.4758967701,
488 -0.0105593596, -0.0013988296, +0.0203939520, +0.0327833831, +2.0266811084, ...
      +1.0930287679, +11.6426431942, +1.5750210620,
```

```
489 -0.0023416459, -0.0009640527, +0.0380591700, +0.0126164179, +0.6753549493, ...
      +0.4758967701, +1.5750210620, +0.9856128248;
490     gaussian[3].setParameters(Sigma, Mu);
491
492     // Gaussian Cluster 4
493     Mu << +0.0232819231, +0.0000612103, -0.1881124599, -0.0096295852, ...
            -2.8940791882, +0.1188011391, +6.5086659061, -3.8651924874;
494     Sigma << +0.0000353101, +0.0000000210, +0.0002721820, +0.0000157467, ...
            -0.0037332421, +0.0022771690, -0.0108205595, -0.0003224990,
495 +0.0000000210, +0.0000120495, +0.0000006333, +0.0000065318, -0.0001958697, ...
      -0.0003073166, -0.0002708839, -0.0008997818,
496 +0.0002721820, +0.0000006333, +0.0043026363, -0.0001656743, -0.0523658898, ...
      +0.0475822501, -0.1910683135, +0.0148711681,
497 +0.0000157467, +0.0000065318, -0.0001656743, +0.0005486006, -0.0008305927, ...
      -0.0055465039, +0.0100352977, -0.0119490947,
498 -0.0037332421, -0.0001958697, -0.0523658898, -0.0008305927, +1.1481597173, ...
      -0.5727291788, +2.7731124141, +0.1139686996,
499 +0.0022771690, -0.0003073166, +0.0475822501, -0.0055465039, -0.5727291788, ...
      +0.8798638368, -2.2315180461, +0.4628799761,
500 -0.0108205595, -0.0002708839, -0.1910683135, +0.0100352977, +2.7731124141, ...
      -2.2315180461, +9.5366756554, -0.6165476415,
501 -0.0003224990, -0.0008997818, +0.0148711681, -0.0119490947, +0.1139686996, ...
      +0.4628799761, -0.6165476415, +0.8224787082;
502     gaussian[4].setParameters(Sigma, Mu);
503
504     // Gaussian Cluster 5
505     Mu << -0.0364319909, -0.0016036845, +0.0003083495, +0.0005350984, ...
            -0.0584604707, +0.1086493185, +0.0597199115, -0.0276415630;
```

```
506     Sigma << +0.0000115808, +0.0000000214, -0.0000004155, -0.0000001961, ...
            -0.0001301561, +0.0001168251, -0.0000072066, -0.0000036862,
507 +0.0000000214, +0.0000103843, -0.0000001295, +0.0000000497, -0.0000307118, ...
        +0.0000594705, -0.0000022583, +0.0000007575,
508 -0.0000004155, -0.0000001295, +0.0000154877, +0.0000008068, +0.0000125632, ...
        +0.0003081702, +0.0000252619, +0.0000134059,
509 -0.0000001961, +0.0000000497, +0.0000008068, +0.0000134891, +0.0003317810, ...
        +0.0000293507, +0.0000151944, -0.0000015038,
510 -0.0001301561, -0.0000307118, +0.0000125632, +0.0003317810, +0.4592063629, ...
        -0.4040184858, +0.0034636019, -0.0000458399,
511 +0.0001168251, +0.0000594705, +0.0003081702, +0.0000293507, -0.4040184858, ...
        +1.0897278872, -0.0010868137, +0.0001450421,
512 -0.0000072066, -0.0000022583, +0.0000252619, +0.0000151944, +0.0034636019, ...
        -0.0010868137, +0.0011218522, +0.0001129399,
513 -0.0000036862, +0.0000007575, +0.0000134059, -0.0000015038, -0.0000458399, ...
        +0.0001450421, +0.0001129399, +0.0002047654;
514     gaussian[5].setParameters(Sigma, Mu);
515
516     // Gaussian Cluster 6
517     Mu << +0.0300156395, +0.0017003014, -0.0126997933, -0.0494809693, ...
            -2.2552491438, +0.2568230751, -5.4469255509, -2.3938466789;
518     Sigma << +0.0000181826, -0.0000009540, +0.0000040073, -0.0000098032, ...
            +0.0011102769, +0.0008219916, -0.0022539276, +0.0008131972,
519 -0.0000009540, +0.0000126472, +0.0000224427, -0.0000076794, -0.0004686015, ...
        -0.0011879195, -0.0043423538, -0.0008892169,
520 +0.0000040073, +0.0000224427, +0.0010900515, -0.0003066255, -0.0026831812, ...
        -0.0277402213, -0.1631539609, +0.0015995042,
521 -0.0000098032, -0.0000076794, -0.0003066255, +0.0004383396, -0.0023626508, ...
        +0.0090217342, +0.0638024483, -0.0041067225,
```

```
522 +0.0011102769, -0.0004686015, -0.0026831812, -0.0023626508, +0.5807803897, ...
      +0.3991951090, +0.9164764304, +0.4433580730,
523 +0.0008219916, -0.0011879195, -0.0277402213, +0.0090217342, +0.3991951090, ...
      +1.2484215420, +4.7650506157, +0.3733543745,
524 -0.0022539276, -0.0043423538, -0.1631539609, +0.0638024483, +0.9164764304, ...
      +4.7650506157, +29.7013008214, +0.2232800980,
525 +0.0008131972, -0.0008892169, +0.0015995042, -0.0041067225, +0.4433580730, ...
      +0.3733543745, +0.2232800980, +0.6453046992;
526     gaussian[6].setParameters(Sigma, Mu);
527
528     // Gaussian Cluster 7
529     Mu << -0.0114078415, -0.0016551774, +0.0352180359, +0.0039512788, ...
          +4.0424747609, +0.1919234205, +0.6723692435, +0.5875883502;
530     Sigma << +0.0000295482, -0.0000009939, +0.0001280147, +0.0000056635, ...
          -0.0010365298, +0.0002899667, -0.0046472047, +0.0010179307,
531 -0.0000009939, +0.0000125656, +0.0000349825, +0.0000004376, -0.0000850415, ...
      -0.0002200252, -0.0011491421, -0.0003958121,
532 +0.0001280147, +0.0000349825, +0.0169485611, +0.0005734190, +0.0030482416, ...
      -0.0104714552, -0.5978675218, +0.0831543625,
533 +0.0000056635, +0.0000004376, +0.0005734190, +0.0002971980, -0.0010249789, ...
      +0.0008836254, -0.0214438845, +0.0014525449,
534 -0.0010365298, -0.0000850415, +0.0030482416, -0.0010249789, +0.3832477265, ...
      -0.0403294563, -0.1072918374, +0.0747422653,
535 +0.0002899667, -0.0002200252, -0.0104714552, +0.0008836254, -0.0403294563, ...
      +0.1145825176, +0.3616200548, +0.0135969752,
536 -0.0046472047, -0.0011491421, -0.5978675218, -0.0214438845, -0.1072918374, ...
      +0.3616200548, +21.4115240506, -2.9702581309,
537 +0.0010179307, -0.0003958121, +0.0831543625, +0.0014525449, +0.0747422653, ...
      +0.0135969752, -2.9702581309, +0.5990486867;
```

```
538     gaussian[7].setParameters(Sigma, Mu);
539
540     // Action manifold 0
541     gmm[0].add(gaussian[2], +0.1738025341);
542     gmm[0].add(gaussian[1], +0.0817487464);
543     gmm[0].add(gaussian[7], +0.1514432086);
544     gmm[0].add(gaussian[0], +0.1569450501);
545
546     // Action manifold 1
547     gmm[1].add(gaussian[0], +0.1569450501);
548     gmm[1].add(gaussian[3], +0.1070950886);
549     gmm[1].add(gaussian[4], +0.1163935355);
550     gmm[1].add(gaussian[6], +0.0892919712);
551     gmm[1].add(gaussian[2], +0.1738025341);
552
553     // Action manifold 2
554     gmm[2].add(gaussian[5], +0.1232798656);
555     gmm[2].add(gaussian[1], +0.0817487464);
556     gmm[2].add(gaussian[7], +0.1514432086);
557     gmm[2].add(gaussian[0], +0.1569450501);
558
559     // Action manifold 3
560     gmm[3].add(gaussian[2], +0.1738025341);
561     gmm[3].add(gaussian[1], +0.0817487464);
562     gmm[3].add(gaussian[7], +0.1514432086);
563     gmm[3].add(gaussian[5], +0.1232798656);
564
565 }
566
```

```
567 // NEURAL NETWORK PARAMETERS
568 void loadNeuralParameters(NeuralParameters &nnet) {
569
570     Eigen::MatrixXd IW1_1(20,5);
571     IW1_1 <<
572 +1.1272713820, +1.4483434349, +1.4798731365, -1.2650600278, +0.0376172552,
573 -0.0015228229, -1.3368147143, +0.1549840877, +1.8388297191, -1.4260773229,
574 -0.8447724200, +1.1907772558, -0.6334072692, +1.2181699021, -1.9414962327,
575 +0.6386844270, +1.4243229652, +1.4002766337, +1.4251342888, -0.2659449502,
576 +1.9574400483, +1.0170226257, +0.9464043280, -1.4890921165, -0.0066465925,
577 -0.2919460975, -0.4551736979, -1.8414253944, -1.6172528424, +0.4528228566,
578 -1.7590294586, +0.8459957959, -1.0645625627, +1.1527458081, +0.6408687055,
579 -1.5224734712, -1.4959813725, +1.4475619166, -0.5147648616, -0.2196358089,
580 -1.7561089311, -0.8268374497, +1.5636490107, +0.0069261952, +0.3636326877,
581 +1.4909249374, -0.2530025418, -1.4077805875, -0.7373068992, -1.1007930627,
582 +0.1389221110, +1.3797585084, -0.8184740753, +1.7945509937, -0.8247545573,
583 +1.4334708930, -1.4752189306, -0.5869336479, +0.2053023651, +1.4027959128,
584 -1.1810402833, +1.8713268265, -0.4341893333, +0.9301144612, -1.6443808686,
585 -1.0418743034, +0.8643890686, -1.4197831648, -1.5629707638, -0.9044177950,
586 +0.7743594144, +1.8163240470, +1.3147578652, -0.0894352748, -0.9826283459,
587 +1.6226402628, -0.5822228342, +0.1019209118, -1.8495724152, -0.1049317026,
588 -0.6677266959, -1.1530199643, +1.1023004041, +1.0923669070, -1.8388110093,
589 -1.3845748526, -0.0190059539, -1.7018601997, -0.1134547669, -1.6535520088,
590 +1.0324780886, +0.9120917949, +1.3550804773, -1.4945967541, +0.0120422713,
591 -1.2301371529, +1.5694802330, +0.7521485874, -1.5977214973, -1.3155445261;
592
593     Eigen::MatrixXd LW2_1(4,20);
594     LW2_1 <<
```

```
595 +0.3702055372, -0.0987564840, +0.5203823598, -0.3715614118, -0.6599985324, ...
      -1.0060804938, -0.6284404684, +1.0070595848, +0.2186204836, ...
      -0.6307081475, -0.1483320342, -0.5949574983, -0.8412696928, ...
      +0.0820928176, +0.7376718429, +0.5849039707, +1.4243340978, ...
      -0.7999410225, -0.2517523842, +0.7049756359,
596 +0.1615406145, -0.5500376792, -0.0108987095, -0.6238020416, -0.0063900866, ...
      +0.3108123873, +0.0964277113, -0.3471138656, +0.0291445487, ...
      +0.9927196528, +0.9780966409, +0.0195756164, +1.6516602781, ...
      +0.4283434037, -0.0464101915, +0.5880955324, +0.1900524370, ...
      +1.0007240537, -0.8339809903, +0.7201648010,
597 +0.1048335205, -0.4649820109, -0.8276561023, +0.9014955233, -0.3082770019, ...
      -0.9623777566, +0.1478708231, +0.7523033706, -0.2155701992, ...
      -0.3241601636, +0.1456857073, +1.3768812429, -0.6575794512, ...
      +0.7871594954, -1.4006718413, +0.7562368026, -0.4588550236, ...
      -0.5136109726, -0.1830625102, +0.4165695587,
598 +0.8586897816, -0.3889116987, -0.1343940977, -0.1337913278, +1.0266065692, ...
      -0.9221548393, +0.4849482358, +0.2305864694, +0.6152245239, ...
      +0.0240164837, -0.9246781985, +0.2535257913, -1.0361457603, ...
      -0.6141530122, +0.9280566465, -0.7221395161, +0.5572947638, ...
      -0.5189538050, -0.1320341708, -0.7768454126;
599
600     Eigen::VectorXd b1(20);
601     b1 <<
602 -2.3281955606, +2.0600005326, +1.7698377704, -1.7442312172, -1.4758084326, ...
      +1.0924517959, +0.7754784427, +0.4226114027, +0.4951496140, ...
      -0.1222771767, +0.1294832944, +0.3757745995, -0.2960332509, ...
      -0.7936540501, +1.1976989305, +1.5241930128, -1.5704499058, ...
      -2.0722308975, +2.4014467643, -2.3255712010;
603
```

```
604     Eigen::VectorXd b2(4);
605     b2 <<
606 +0.1594944711, +0.6002299289, +0.2674196203, -0.2231267414;
607
608     Eigen::VectorXd gain(5);
609     gain <<
610 +0.2666005968, +2.0000000000, +2.0000000000, +2.0000000000, +2.0000000000;
611
612     Eigen::VectorXd xoffset(5);
613     xoffset <<
614 -0.0000068435, +0.0000000000, +0.0000000000, +0.0000000000, +0.0000000000;
615
616     double ymin = -1.0000000000;
617
618     // Write to the nnet parameter struct
619     nnet.il_ymin   = ymin;
620     nnet.il_xoffset = xoffset;
621     nnet.il_gain   = gain;
622     nnet.l1_b      = b1;
623     nnet.l1_W      = IW1_1;
624     nnet.l2_b      = b2;
625     nnet.l2_W      = LW2_1;
626
627 }
628
629 #endif
```