Intelligent Behavior in Autonomous Agents through Optimization and Learning

Kendall Lowrey

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

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Emanuel Todorov, Chair

Dieter Fox

Sham Kakade

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Intelligent Behavior in Autonomous Agents
through Optimization and Learning

Kendall Lowrey

Chair of the Supervisory Committee:
Associate Professor Emanuel Todorov
Computer Science & Engineering

The dream of intelligent robotics performing useful tasks is predicated on the ability to quickly generate complex behaviors. These systems must be able to dynamically react and adapt to unstructured environments and unexpected situations, all without having full awareness of the complexity of the world around them. These behaviors then cannot be pre-planned with prior insight, and indeed many complex tasks might be difficult for humans to even specify algorithmically. A successful method must balance the ability to perform well quickly with prior information while also learning from experience to improve over time; as additional computing resources are increasingly available these ideas are realizable. In this thesis we present work involving the combination of model free and model based paradigms. The interaction of these methods allows for faster behavior synthesis through trajectory optimization while also incorporating experience discovered through planned exploration to enable more long term planning.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to gratefully acknowledge the inputs and contributions of my doctoral committee, without which I would not nearly think as broadly or critically. Notably my advisor Emo Todorov is a constant source of brilliant insights, useful shortcuts, both programmatically and mathematically, and ever able to incite laughs. My collaborators and fellow members of the Motion Control Lab, Aravind Rajeswaran, Igor Mordatch, Svet Kolev, Vikash Kumar, Yuval Tassa, and Tom Erez, have been endless sources of ideas, references, advice, and simple know-how. Other members of the University of Washington Computer Science & Engineering Department, too numerous to name, have patiently tolerated my numerous questions and outlandish queries, and generally expanded my ability to think, a special thanks to Robert Gens and Mike Chung. Friends outside of the department deserve all the same gratitude, as the trajectory of my life would not be the same without your pushes and/or perturbations.

I would not be who I am today without the love and support of my extended family that has taught me to be willful, and my parents that have contributed so much to my nature, and guided me through with gentle nurture. I see you both in everything I do, and I hope that I have made you proud. Finally, I have a collaborator and partner in life through my wife, Renee Yang, thank you so very, very much; your support, suggestions, and love have made this work entirely possible.

Last but not least, Talisker and Sitka may appear to be hirsute cats, but are truly a wellspring of inspiration in the research topics of motion control, multi-stage problem solving, and creative exploration.

Chapter 1

INTRODUCTION

The notion of an intelligent agent is one that possesses an ability to acquire and apply knowledge and skill towards solving complex tasks. The motivation of research in developing intelligent agents, often embodied as robots, is both the practical assistance in human endeavors as well as the potential insight into how we operate. Human progress has grown through the development of tools that augment our capabilities, and as we attempt more and more complex tasks, we in turn imagine more mechanisms to complement and augment our capabilities; the vision of intelligent systems working alongside humans has existed since written history, but we are at a point where technical capability makes the vision imminently possible.

Unlike more mechanical systems, the notion of *intelligent* systems implies the ability to vary its performance depending on context, and critically, the ability to change performance depending on the past: learning from experience, especially unplanned situations, is a defining characteristic of intelligence. Much like how mechanical systems can provide leverage for physical tasks impossible for humans to perform, can intelligent systems provide a similar lever for tasks that require abstract reasoning, vast amounts of data, and, in the case of robots, also additional physical challenges?

The definition of intelligence is difficult to pin down due in large part to its context dependent nature, leading to numerous challenges in developing intelligent systems. High performing systems have been developed for fixed contexts and settings, such as chess, Go, or even video games [30, 131, 43], both through traditional methods as well as reinforcement learning techniques that allow for an agent to develop its skills through experience. However, results of this level often require a combination of human insight and significant amounts

of experience, along side not insignificant computing power to achieve, and are not quickly adaptable to *new* contexts or changes.

Rather than viewing intelligence as individual or collection of goals to be optimized for, this work prefers to consider the development of systems that exhibit behavior. Rather than being defined by specific tasks to be accomplished, we would like systems that conduct themselves stably while attempting to navigate the interaction between the agent's intent and the dynamics of its setting. We assume that a system that understands this interaction leads to intelligent behavior. This notion is particularly clear in robotic applications where physical systems do not have the luxury of making millions of uninformed trials. If the intent is to perform a task, the robot should ideally grow to understand how the physical environment may respond to their actions and modulate its actions in response.

In the same way that intelligence is hard to define, behavior is a challenge to specify. However, the use of overly specific tasks or goals as surrogates of intelligence or behavior avoids the fact that intelligent behavior in dynamic settings is similarly dynamic, shifting around a continuum of action while still allowing the agent to accomplish its goals. While we may still be far from this vision, both in definition and implementation, this work's primary contribution is the development of a method combining online trajectory optimization with learning from experience that allows agents to perform well initially, but also to improve after gaining experience. The combination of techniques allows for an agent to make useful progress along its intent, and through said progress, better understand the dynamics of its setting to improve.

Trajectory optimization–and state estimation–may be considered by some to be "classical" techniques given their reliance on a model of the dynamics and their long history of application to real world problems. There is still interesting and useful research in this space as attempts are made to extend and generalize these methods to more complicated robotic systems in more complicated scenarios. For example, settings involving a robot making physical contacts such as walking or manipulation of objects are fundamentally difficult to model correctly. Modelled well, however, and an agent can optimize a trajectory of actions that performs tasks

orders of magnitude more efficiently than methods that eschew modelling. It should be noted that since robots are imagined, designed, and made by humans, we can generate accurate models of them.

A modification of trajectory optimization to further increase efficiency is to operate in a Model Predictive Control (MPC) fashion, alternatively receding horizon control, where an optimizer only plans and executes a short time horizon before re-planning. This saves processing time of the optimizer at the cost of a (slightly) less optimal trajectory. Due to the constant replanning, the agent or robot may be very robust to unexpected disturbances, but is limited by both the horizon of planning, and any errors in the dynamics model. Whether due to a limited plan or error in the plan's execution due to model mis-match, these methods generally do not adapt to new information or experience. More concretely, if a trajectory optimizer cannot solve the problem initially, it will not be able to: if a goal is beyond the planning horizon, the agent may not "see" a way to get to the goal.

If model based optimization is on one end of a spectrum, then model free reinforcement learning can be considered on the other end. These methods eschew the prior information of a dynamics model, and instead optimize for a set of rules that guides the agents actions. Whether as a policy or value function, these methods optimize the Bellman equation [16] and have useful expectations of performance. These methods directly optimize the parameters of a the value function or policy by executing sequences of actions in the world and scoring them with a defined reward function. By not being constrained to what a dynamics model may inform, these methods are potentially able to find more optimal trajectories than MPC, at the cost of performing numerous sequences of actions in the world; this is often prohibitive for robotics directly as they may not physically survive the amount of samples. Conceptually, these methods are very elegant as they are purely data driven by the experience of the agent and all performance is derived from experience, even if they require tremendous amounts of experience.

The goal of combining these two ends of a spectrum is to allow for their strengths to mitigate the other's weaknesses: a model based optimizer can continuously re-plan and

perform non-trivial goal directed actions, while a value function incorporates experience to learn long term value that guides the optimizer. This way, the agent may always have a means to accomplish its intent but also improve its abilities over time. We detail this combination of methods, as well as a means to perform goal directed exploration in Chapter 6. The next chapter is an overview of related works, and Chapter 3 discusses a project utilizing model ensembles to improve state estimation. This notion of widening a distribution is additionally used for reinforcement learning performance in Chapter 4 and 5 in both simulated and hardware experiments.

Chapter 2
RELATED WORK

The problem of intelligent behavior discovery touches upon many fields and applications. This chapter will review related works and relevant approaches of trajectory optimization, reinforcement learning, as well as attempts to combine the two and applications to hardware systems.

2.1 Classical Methods

Classically, complex control systems such as robotics has been split into a paradigm of estimation and control [28]. A designer of the system must specify which variables correspond to the state of the system and which are the control inputs, and design algorithms that alternate between estimating the often unobserved hidden state and optimizing for controls that drive the system to desired states. In this section we discuss related works and applications relating to this paradigm.

2.1.1 Estimation

State estimation, in some form, is utilized in almost every robotic platform to allow for observed values from sensors to predict unobserved variables such as the hidden state of the system. While modern techniques may attempt similar control tasks without an explicit estimation step–by mapping observations directly to controls–understanding how estimation techniques work gives insight into the value of good observation features for predicting the uncertain variables from which an agent or robot must react.

In the space of humanoid robotics applications, models are often approximated as an inverted pendulum model that shifts its base during footsteps [61, 136]. This technique was

originally developed to simplify the planning of walking, but similarly extends to estimation. Some model simplification makes the inherent assumption that the robot's movements are quasi-static in its manipulation of the environment, which is a common strategy [21, 154, 97]. This of course drops information about more complicated dynamic actions and motions. Additional information may be provided to the estimator through contact location [24]. As contacts are a critical phenomena of an under-actuated robot, planning trajectories featuring contact and providing this contact information allows an estimator to perform better, but requires prior planning.

Another technique for underactuated robots to split the estimate between root orientation, using a high quality inertial measurement unit, and the joint state of positions and velocities [69, 158, 134, 17]. This results in information loss between the joint configuration and the root, and may also be used in addition to a reduced model. This type of split estimation allows for increased drift as the root has less information to integrate regarding the joint's contacts with the world; this is usually overcome with vision and LIDAR systems providing accurate root position and orientation [44]. In a similar way, [74] is able to track a high degree of freedom human hand in a cluttered environment utilizing information dense vision and contact physics.

2.1.2 Trajectory Optimization

To avoid the difficulty of manual specification of trajectories, modern approaches utilize a model of the system to optimize a trajectory for a given task. Also referred to as planning algorithms, these approaches are increasingly complex, sophisticated, and are able to find success on real physical systems. They are not without certain caveats, however. Many early attempts used simplified models of a system for tasks such as bipedal walking [62, 73, 128] and push recovery [135, 113] that, by design, restrict the robotic agent's movement regime. Despite this, trajectory optimization with simplified models has been demonstrated to work on varied robotic systems such as quadrupeds [116, 23, 48, 56] and bipeds [4, 69].

One method to increase trajectory fidelity is to utilize a more complete dynamics model,

which introduces two problems. First, with higher fidelity dynamics, the chance of the optimization landscape being more highly non-linear increases. Direct trajectory optimization methods [53, 112, 25, 111] are able to incorporate additional information about the dynamics to smooth the optimization and increase performance. Namely Contact Invariant Optimization incorporates information about contact phenomena to plan for long term trajectories [89, 91]. The second difficulty is in the increased computational demand of a higher fidelity model. Model predictive control, alternatively known as Receding Horizon Control is an effective way to trade off less computation for optimality [141, 142, 40, 37, 22, 129, 110] by planning for a short term horizon quickly, and re-planning soon after.

Additional challenges with trajectory optimization based approaches for control includes the requirement for the task to be fully specified through a reward function. Given that the dynamics of the systems may be highly non-linear, the manual specification of a reward function may be very unintuitive: translating the high level symbolic information a human reasons at to a set of state dependent terms and coefficients is nontrivial and can result in unwanted or reduce performance. This problem highlights the fact that trajectory optimization based approaches by themselves do not learn from the data they experience: using a model and trajectory optimization can efficiently lead to good performance, but that performance will be fixed. This thesis offers a method to improve performance from experience and to even make up for limitations in the task specification.

2.2 Deep Reinforcement Learning

Reinforcement learning (RL) concerns the process of allowing an agent to select an action in an environment to maximize some notion of reward or utility [139] that it receives after executing the action. It thus must learn from this interaction how best to maximize reward. Unlike classical control methods, reinforcement learning attempts to provided a reliable framework for the *autonomous* discovery of optimal rewards in numerous domains [147, 11, 35, 143, 60]. A goal in reinforcement learning is to discover a value function or policy function the agent may query to maximize its reward; if this function cannot be pre-specified, approximating

this function is a viable method for the agent to make progress [13, 140, 26, 146, 65, 132]. More recently, Deep reinforcement learning (deepRL) has recently achieved impressive results on a number of hard problems including sequential decision making in game domains [43, 42] by pushing the use of very large function approximators, namely deep neural networks. This success has motivated efforts to adapt deepRL methods for control of physical systems, and has resulted in rich motor behaviors [125, 79]. These methods are generic, automatic, and can utilize unstructured or discontinuous reward functions and still perform well, at the cost of increased sample complexity. Unlike trajectory optimization that can leverage a model of the dynamics to guide the agent's exploration, deep reinforcement learning acquires information through sampling the dynamics directly; this can be time consuming depending on the dimensionality, the environment, and the task.

Recent success of RL [42, 76, 72, 71, 108] has been enabled largely due to engineering efforts such as large scale data collection [43, 42, 108] or careful systems design [76, 72] with well behaved robots. In effect, these approaches attempt to alleviate the difficulty of exploration. In the next section, we discuss works that combine model based optimization with learning.

2.3 Combining Optimization with Learning

While reinforcement learning avoids the need for a model, it can be constrained by the need to successfully explore a high dimensional state space. Trajectory optimization or planning, on the other hand, relies on prior model information to more efficiently explore, but can be hampered by the inability to overcome local solutions. As such there have been a number of attempts to utilize both to overcome each method family's limitations in terms of sample complexity and exploration.

2.3.1 Planning and Learning

If model based planning optimizes with respect to a provided model, then learning should allow for the incorporation of data experienced to improve optimizer performance. Even

without learning, there are methods that attempt to boost a planner's performance through additional information.

Combining elements of planning and search with approximate value functions has been explored in discrete game domains [144, 131, 3] where a Monte Carlo Tree Search (MCTS) planner is informed by information learned through experience. Alternatively, using prior data to guide the search process in continuous MCTS without explicitly learning a value function has also been explored [117]. Related to this, [6] uses an offline trajectory library for action selection in real-time, but do not explicitly consider learning parametric value functions. RTDP [14] considers learning value functions based on states visited by the agent, but does not explicitly employ the use of planning. [160] consider the setting of learning a value function to help model predictive control (MPC), and found the contribution of value functions to be weak for the relatively simple tasks considered in their work. Approaches such as cost shaping [95] can also be interpreted as hand specifying an approximate value function, and has been successfully employed with MPC [141]. However, this often requires careful human design and task specific expertise. An alternative set of approaches [123, 77, 90, 138] use local trajectory optimization to generate a dataset for training a global policy through imitation learning. These approaches do not use MPC at runtime, and hence may often require retraining for changes in tasks or environment. Furthermore, results from this line of work have been demonstrated primarily in settings where trajectory optimization alone can solve the task, or use human demonstration data.

2.3.2 Planning and exploration

Exploration is a recognized and important problem in reinforcement learning. The importance of having a wide and relevant state distribution has been pointed out in numerous prior works [93, 10, 121]. Strategies such as ϵ-greedy or Gaussian exploration have recently been used to successfully solve a large number of problems where the reward function presents a sufficient signal for an algorithm to follow. As the reward becomes sparse or heavily delayed, such strategies become intractable in high-dimensional settings. Critically, these approaches

perform exploration on a per time-step basis, which can lead to dithering and prevents efficient exploration. Parameter-space exploration [109, 46] methods do not explore at each time step, but rather generate correlated behaviors based on explored parameters at the start. However, such approaches do not consider exploration as an intentional act, but is rather a deviation from a well defined objective for the agent. Deep exploration strategies [101] sample a value function from the posterior and use it for greedy action selection. Approaches based on notions of intrinsic motivation and information gain [33, 133, 55, 103, 15] also explicitly introduce exploration bonuses into the agent's reward system. However, such approaches critically do not have the element of planning to explore: the agent may not actually reach regions of high predicted reward because it does not know how to get there. Our work is perhaps closest to the $E3$ framework of [64], which considers altered MDPs with different reward functions, and executes the optimal action under that MDP. However solving these altered MDPs is expensive and their solution is not reusable. Leveraging trajectory optimization and planning, on the other hand, can quickly solve for local instance-specific solutions in these MDPs.

2.4 Model Free vs Model Based

While model free methods may find novel solutions without prior knowledge, their efficiency often prevents use on real robotic systems [66]. On the other hand, providing an accurate model presents its own challenges [58, 96] that must be overcome. An approach to mitigate the problem of a mis-matched model can be found in the process of system identification for offline training, and deploying the subsequently learned policy to the hardware system. While there have been some cases of reinforcement learning directly happening with hardware in the loop [51, 36, 107], the latent space dimensionality was often small which reduces training time, or human demonstrations and/or parameterized learning was utilized to accelerate exploration [32, 159, 67, 119, 85]. Instead, the use of a simulated model for training before application to hardware is referred to as 'sim to real', and applies to both model based and model free methods. Methods that do not rely on online adaptation of model parameters

[88, 102, 153, 75, 122, 83, 12, 118] avoid the problem of an imperfect model by increasing model diversity during training, referred to as 'domain randomization' or 'model ensembles' [104, 31, 87, 81, 34, 1]. In short, there are a number of methods to mitigate the issue of model mis-match, even when deploying a policy to robotic hardware; in this thesis we demonstrate how accurate system identification along with model ensembles can work for even very difficult to control systems.

While model free methods, with the proper consideration, be used to train behavior in simulation before deploying to hardware, the use of a model can still increase efficiency to a significant degree, assuming the task is amenable to trajectory optimization. As a representative number, [125] report approximately 5 days of agent experience and 128 CPU core hours for solving tasks such as a humanoid agent getting up from the ground. In contrast, trajectory optimization can perform the same task in real-time [141] with a full dynamics model, albeit with much more computational overhead at runtime. Recently, policy gradient methods were also used for in-hand manipulation tasks [98], where 3 years of simulated experience and 500 CPU hours were used for object reorientation tasks. In this work we present a method that performs the same tasks 12 and 1 CPU hour, respectively. The assumption of a dynamics model is a caveat, of course, but in an alternate line of work, internal models have been used for variance reduction purposes in model-free RL [45, 29]. Related to this, [9] consider learning an internal model for discrete action domains and use short horizon Monte Carlo Tree Search for planning. Similarly, [94] learn a dynamics model in simple continuous control tasks and use a random shooting MPC method for action selection. This is all to say that a dynamics model can also be learned online as well that is exploited in the same way to speed up behavior discovery over purely model free methods. The trade-off between model free and model based methods may only be determined on a case by case basis, but this work entails a method that combines features of both that complements each method's weaknesses.

Chapter 3

REAL-TIME STATE ESTIMATION WITH MULTI-CONTACT DYNAMICS

This chapter discusses a real-time state estimator applicable to whole-body dynamics in contact-rich behaviors, based on the Unscented Kalman Filter (UKF) [59], with modifications that proved essential in the presence of strong contact non-linearities combined with unavoidable model errors. Instead of the standard UKF weighting scheme, we use uniform weighting which is less susceptible to samples that violate unilateral constraints. We also develop a rich model of process noise including noise in system parameters, control noise, as well as a novel form of timing noise which makes the estimator more robust. The method is applied to an enhanced Darwin robot in walking as well as pseudo-walking while lying on the floor. Estimation accuracy is quantified via leave-one-out cross-validation, as well as comparisons to ground-truth motion capture data which is not used by the estimator.

3.1 Introduction

Accurate state estimation is a prerequisite for developing robust and efficient feedback controllers for modern robots. Such robots have high-dimensional dynamics that cannot be modeled exactly, and are subject to strong non-linearities (particularly due to contact phenomena) which make estimation challenging. Our goal is to develop a general estimator that works in real-time and yields results that can be safely fed into a whole-body feedback controller.

The starting point of our work (as well as almost every other approach to estimation) is recursive Bayesian inference. However, with the exception of a few special cases, we can not represent the true Bayesian posterior. Thus, we must resort to some approximation, seeking

a favorable trade-off between approximation accuracy and computational efficiency. The two most popular approximations are the sample approximation used in particle filters, and the multivariate Gaussian approximation used in Extended Kalman Filters (EKF) and Unscented Kalman Filters (UKF). Particle filters have the theoretical advantage that in the limit of infinitely many samples, they converge to the true posterior, but the number of samples required for high dimensional models many samples proves difficult for real-time computation.

Gaussian approximations on the other hand can become very inaccurate and cause filter divergence. The UKF is known to alleviate this problem compared to the EKF, however it comes without any stability or optimality guarantees for general nonlinear systems, and can diverge in practice. Filter divergence if often caused by over-confidence, which in turn reflects insufficient uncertainty in the dynamics or sensor models. Indeed we have found that the traditional UKF in our setting usually diverges. In order to make the filter stable and accurate, we introduce two modifications. First, we weight the samples computed by the UKF uniformly, instead of using the standard weighting schemes derived from considerations of higher-order accuracy. Uniform weighting increases robustness to samples that violate contact constraints and require the filter to postulate very large external force in order to explain them. Second, we develop a stochastic dynamics model with several noise sources: system parameter uncertainty obtained from system identification; timing noise that models the fact that sensor measurements accumulate during the time step; and control noise that models the fact that our actuator model is not perfect and there is some backlash. We show empirically that including all three noise sources results in more accurate estimation than including any two, any one, or none of them.

Another important concern is real-time performance. Even though the UKF is much faster than a particle filter, it still requires $2N + 1$ evaluations of the system dynamics where N is the state space dimensionality. Even a relatively simple humanoid such as the Darwin robot we use here has 26 DOFs (6 of them are unactuated torso DOFs) resulting in $N = 52$. Thus we need 105 dynamics evaluations per update, or in other words, our simulator has to run 100 times faster than realtime. We achieve this partly through parallel processing.

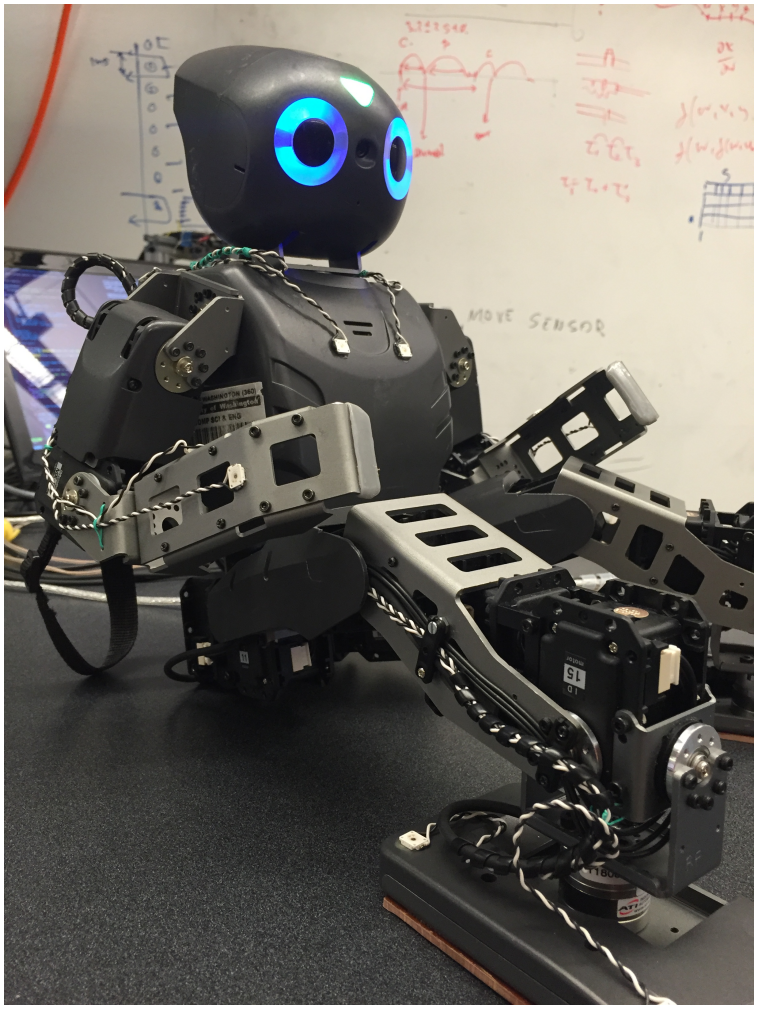


Figure 3.1: A picture of the Darwin-OP robot used in experiments.

The remaining speedup reported here is due to the efficiency of the MuJoCo simulator we use, and the way we warm-start the iterative contact solvers (which are found in all modern physics simulators, and are usually the computational bottleneck). Our estimator is able to handle the whole-body dynamics of the robot, and perform an update in about 7 msec on a laptop processor. Importantly, we hide most of that latency in the time when the computer is waiting for the next sensor data to arrive. This is possible because the UKF has the usual predictor-corrector form, and the predictor step does not require the new sensor data.

3.2 Our Method

We define the problem as follows: we wish to estimate a robotic system's state $x_k = [q_k; v_k]$ where q_k and v_k are the generalized configuration (positions) and velocities of the degrees of freedom of the system at some time k. We utilize an Unscented Kalman Filter (UKF) to propagate sample points through the non-linear state transition function. We briefly describe our UKF as follows. We wish to estimate our state and state covariance matrix at some time k given our previous state at time $k-1$, which we denote as $\hat{x}_k$ and P_{ks}.

We generate a set of $2N+1$ sigma points–which we denote as χ_i for the ith point–where N is the size of our state vector x_{k-1}. Our sigma points are the set of points perturbed from the original as below; we may drop the $k-1$ term for clarity.

$$\begin{aligned}
\chi_0 &= \hat{x}_{k-1} \\
\chi_{i=1:N} &= \hat{x}_{k-1} + (\sqrt{(N+\lambda)P_{k-1}})_i \\
\chi_{i=N+1:2N} &= \hat{x}_{k-1} - (\sqrt{(N+\lambda)P_{k-1}})_{i-N}
\end{aligned}$$

where

$$(\sqrt{(N+\lambda)P_k})_i$$

is the ith column of the matrix square root (we use a Cholesky Decomposition), and λ is a constant. We then propagate each χ through the state transition function f which accepts the sigma point and controls u to find our state at time $k+1$.

$$\chi_{i,k} = f(\chi_{i,k-1}, u_k)$$

The first step in a Kalman process is the prediction. In a UKF, we sum each sigma point χ_i to create our *apriori* state $x_{k|k-1}$ and covariance $P_{k|k-1}$ estimate:

$$\hat{x}_{k|k-1} = \sum_{i=0}^{2N} W_i \chi_i$$
$$P_{k|k-1} = \sum_{i=0}^{2N} W_i [\chi_i - \hat{x}][\chi_i - \hat{x}]^T$$

where W_i are the chosen weights applied to each sigma point.

Similarly the vector of sensors measurements $\gamma_{i,k}$ can also be gathered through this Unscented transform process where the observation function h is used instead of the state transition function f.

$$\gamma_{i,k} = h(\chi_i)$$

These predicted sensor measurements and sensor covariance are also combined with a weighted sum, in addition to the cross covariance matrix.

$$\hat{z}_k = \sum_{i=0}^{2N} W_i \gamma_i$$
$$P_{z_k,z_k} = \sum_{i=0}^{2N} W_i [\gamma_i - \hat{z}_k][\gamma_i - \hat{z}_k]^T$$
$$P_{x_k,z_k} = \sum_{i=0}^{2N} W_i [\chi_i - \hat{x}_{k|k-1}][\gamma_i - \hat{z}_k]^T$$

These values along with the observed sensor data z_k are used to calculate the Kalman gain matrix K_k and subsequently calculate the *a posteriori* state and covariance estimate.

$$K_k = P_{x_k,z_k} P_{z_k,z_k}^{-1}$$
$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(z_k - \hat{z}_k)$$
$$P_k = P_{k|k-1} - K_k P_{z_k,z_k} K_k^T$$

3.2.1 UKF Weights

While there are many techniques to choose the weights W_i used in the Unscented transform, we eventually settled upon even weighting as the most stable for our use case. Traditionally,

any weights may be chosen so long as $\sum W_i = 1$, including negative weights in many use cases [59]. However, we found that the use of weights less than 0 or greater than 1 exacerbate unrealistic forces induced by the discontinuous state constraints that a robot experiences through contact. Said another way, if a particular sigma point χ_i is in violation of the contact constraints, there is no particular rule to prevent it from inducing bad data into the prediction and correction of the UKF except by having weights $0 < W_i < 1$ where $W_i = 1/N$.

We attempted to preserve the traditional weighting scheme by adjusting those sigma points known to be in violation of contacts by using an adaptive line search style re-weighting of the perturbance $(\sqrt{(N+\lambda)P_k})_i$ whilst also adjusting the particular weights W_i according to [152], but this proved to not be as successful as the even weighting scheme.

3.2.2 Modeling Error & Process Noise

Modeling error is an ever present difficulty in accurately predicting robotics. Simple errors can manifest themselves as constant biases in the system's state transition function $x_k = f(x_{k-1}, u_k) + \epsilon$ such as Gaussian noise; these can be readily reduced in standard system identification procedures. More systematic errors, especially those involving modeling the physical phenomena of contact can dramatically affect state transitions with even minor changes.

Additionally, the real world also suffers from the affliction of process noise, which is any difficult-to-model effects that affect the state vector. While known sources of process noise can be estimated along-side the state in an augmented UKF [157], complicated physical systems such as under-actuated robotics may not be able to quantify this relationship. Thus we have two sources of discrepancy between our modelled state transition function: the process noise and modeling error. To accurately estimate a robot's state we need to overcome these sources of error.

We may represent an ideal state transition function as f_{ideal}, which is a function of the

state and controls that is dependent on our model. We can rewrite it as such:

$$x_k = f_{ideal}(x_{k-1}, u_k) = f_{model}(x_{k-1}, u_k) + q_k(x_{k-1}, u_k)$$

where f_{model} is our known, modeled state transition and q_k is our unknown process noise function that may be time varying. We first make the observation that forces applied along the degrees of freedom of the robot can mimic the process noise or overcome the difference between real and modelled state transitions, similar to the changes induced by q_k. Other sources of error could be changed through the modification of the underlying model governing f_{model}, if only we knew better model parameters.

These two ideas coupled with the UKF's sample points gives us the opportunity to sample from these sources of noise:

$$\chi_{i,k} = f(\chi_{i,k-1+N_k}, u_k + n_u, M + n_m)$$

where n_u and n_m are control noise and model noise respectively.

The parameters that we add noise to are the control vector, model mass parameters, and the time index as zero mean Gaussian noise. The time aspect is inspired by the knowledge that our sensor readings are not instantaneously time synced across all sensors coming from hardware. As such the variance of the Gaussian noise is the variance of the dt (delta timestep) of the collected sensor data. The model mass noise parameter was determined from a system identification procedure of measuring the robot mass in different configurations to discover any measurement discrepancies. This mass noise is distributed randomly among the different body components of the model whilst still normalizing to the expected mass of the robot (2.97 kg). Mass, of course, is an underlying parameter that effects all aspects of the state transition, and from prior work [87], we found it to be critical in generating robust trajectories. Finally, varying the controls is most analogous to changing the degree of freedom forces to mimic process noise, but also helps to overcome the difficulty of modeling backlash and deadbands found in geared, position controlled motor systems.

In the end, these noise sources serve two functions. One is to approximate the stochastic process noise and overcome the modeling error inherent in real world robotics, and secondly,

Table 3.1: Approximating Process Noise

Noise Source	Standard Dev.
Time	0.00096 seconds
Mass	0.012 kg
Control	1e-4 radians

to encourage our UKF formulation to develop appropriately descriptive state covariance matrices. Although these parameters could be included in the x_k state vector of positions and velocities as additional state terms, sampling from a distribution for the noise does not increase computational load (as we do not have additional sigma points) as well as encourage the state covariance matrix to richly describe the state variability at time k. If all the samples in a simulator integrate forward in time with the same deterministic state transition function, the covariance matrix hardly provides useful information. While process noise is frequently sampled and added to each sigma point's state vector, this approach leads to constraint violations which can contribute additional inaccuracies to the estimate.

It should be noted that adding this kind of modelling noise is not apparent in estimation techniques that are not sampling based. Control and time noise are able to be included in frameworks such as a Moving Horizon Estimator or EKF, but it is less obvious to include mass noise when your framework depends on state and control linearization to calculate Jacobian matrices.

3.2.3 Sensor Noise

Another critical detail of any estimator is quantifying the sensor noise. As highly accurate sensor hardware is prohibitively expensive, we overcome sensor noise through a combination of system identification and sensor weighting. Gathering statistics on sensor noise such as

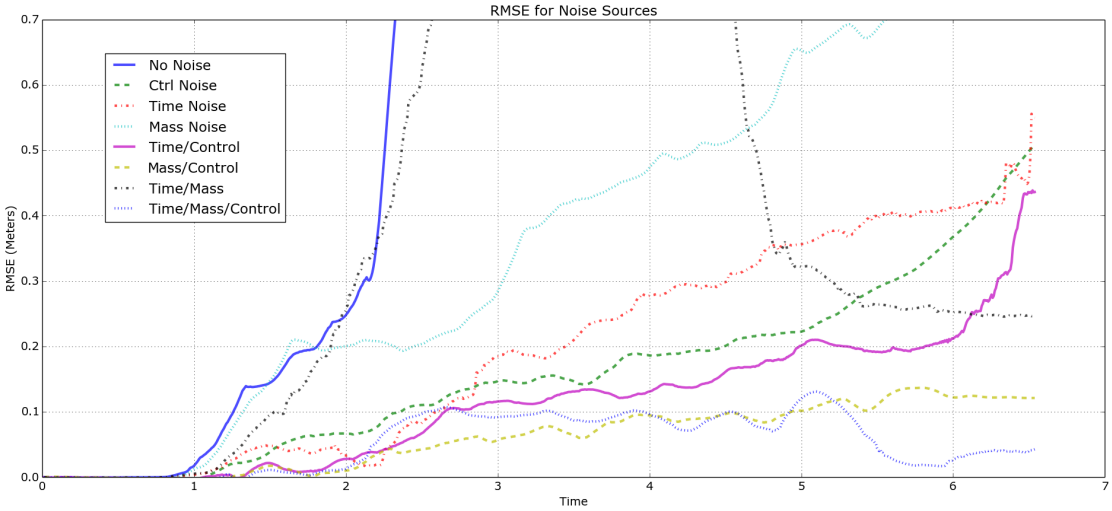


Figure 3.2: Shown is the Root Mean Squared Error for a walking trajectory with the estimator utilizing different sources of noise. Lines that leave the plot area usually have experienced an estimation divergence. The Phasespace markers used to calculate the RMSE were located on the Darwin-OP torso to isolate any drift, as limb deviations could contribute to high RMSE without causing drift. It should be noted that the estimation performed here did not use the Phasespace data in the estimation loop; drift would be nearly inevitable without some similar position based sensor.

mean and variance is standard procedure for any robotics application. Using these values, we have a baseline for the diagonal of the sensor covariance matrix P_{z_k,z_k}. This, in effect, tells the estimator how much to trust the sensor readings from the robot hardware, and is a set of tunable values acting as sensor weights. Some values need to be adjusted due to large dynamic range of the sensed values during operation, as well as the sensitivity of the estimator to some sensors. For example, force and torque sensors can capture highly discontinuous events that can cause an estimate to over-correct, while a gyroscope is an intrinsic sensor centered at zero.

Table 3.2: Assumed sensor noise variance. We first measured the sensor variance from collected data, and adjusted these values empirically to benefit estimator performance.

Accelerometer	1e-4
Gyroscope	1e-6
Force	1e-3
Torque	1e-2
Joint Position	1e-7
Joint Velocity	1e-5

3.3 Practical Considerations

Estimators need to provide data accurately and quickly to allow for successful robot operation of dynamic controllers. As such, we utilize the UKF over other means due to the performance benefits of a specific, controlled sample set – unlike a Particle Filter – while maintaining accuracy in dynamic situations such as making and breaking contact. In the following section we describe a few methods used to achieve real-time performance, which we define here as completing all estimator computations within the time between sensor updates.

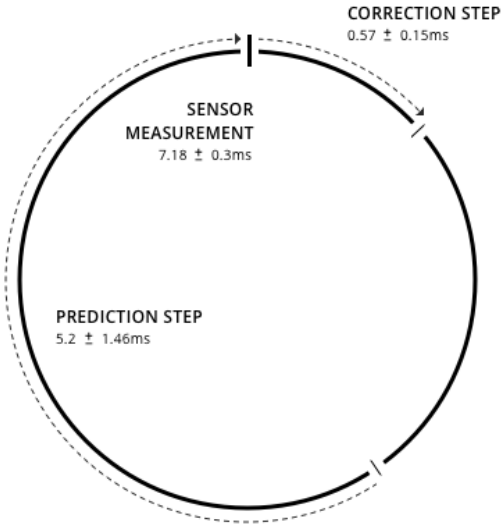


Figure 3.3: Our estimation loop consists of a prediction phase and correction phase that must happen within the sensor measurement loop. As the sensing happens consistently, we can perform our prediction step before receiving sensor data (where normally we would use the measured time delta to integrate in our state transition), which leaves the Kalman correction step to happen after the sensors have provided data. As our process loop happens to be fast enough, we have time to evaluate from a feedback controller based on the calculated estimate between the correction and prediction steps.

3.4 Robotic Platform

Our simulation environment is based on the MuJoCo physics engine, which allows for fast evaluation of dynamical systems even with contact [150]. While MuJoCo is nominally designed to allow for fast finite differencing, the UKF style of estimator is computationally similar to a finite difference procedure. Our UKF needs to evaluate $2N + 1$ sample points centered on one centroid point through forward dynamics. As such the accelerations calculated for the centroid point can be used to warm-start the dynamics evaluations for the rest of the sigma points, reducing computational overhead through data reuse.

Table 3.3: Estimator Timing

Machine	Prediction Time (ms)	Correction Time (ms)
Desktop	5.2 ± 1.46	0.57 ± 0.18
Laptop	6.74 ± 1.22	0.66 ± 0.21

Further performance improvements arise from full utilization of modern CPU architectures. As each sigma point is evaluated independently from the others, our computations are highly parallelizable which we take advantage of through multithreading the dynamics evaluation. Finally, because our robotic platform's sensor updates have consistent timings, we opt to complete the prediction step of the Kalman Filter *before* the arrival of new sensor data, allowing a new state estimate to be completed in less than 1 millisecond after new data. This is intended to benefit future dynamical control efforts by being able to utilize our estimated state as soon as possible after new sensor data is received.

The Darwin-OP humanoid biped produced by Robotis Ltd. is a 26 degree-of-freedom robot – 3 root positions, 3 root orientations, and 20 actuated joints. Each actuator is a MX-28 servo motor with position measurement of 12-bit resolution over 2π radians that is position controlled. Prior work utilizing this platform led us to encounter a number of technical difficulties in using the platform as is, as such we modified the original hardware to increase sensing capability and compute performance as follows.

The original on-board low-power x86 CPU was removed to make way for a higher performance inertial measurement unit (IMU), the PhidgetSpatial High Resolution, which connected to an installed USB hub alongside the original Darwin-OP motor sub-controller board. Additionally, the original force resistive sensing feet were replaced with two ATI Nano-25 6-axis force/torque (F/T) sensors. These F/T sensors need to be connected to large signal amplification boxes off the robot; as such their cables were routed to the back of the

Darwin-OP torso and bundled with the USB cables necessary to communicate with the IMU and sub-controller. These amplification boxes were then routed to a analog to digital signal converter which also communicates over USB. Finally, 16 Phasespace Impulse IR tracking markers were installed on the torso and limbs of the robot to serve as ground truth tracking for estimator verification. These markers needed to be powered by a large wireless pod which was also off the robot platform.

As the motors, IMU, and F/T sensors all communicate over USB, we have the option to link our modified Darwin-OP platform to any computer system we would like. For our purposes we developed and tested our estimator on both a desktop with an Intel Core-i7 5930k CPU as well as a laptop containing a Core-i7 4710HQ processor. While our estimator's evaluation proved quick enough on the laptop, we also tested on the larger desktop processor to gauge additional performance improvements. These details are discussed in section 3.3.

3.5 Experiments

We tested our estimator on data collected from our robotic platform through an open-loop, fast, ZMP-style walking (software provided by the manufacturer) across a flat surface as well as the same walking process while the robot was lying face down. The walking takes approximately three steps every two seconds. The face down test was to examine reduced sensor usage – as the robot was not standing on its contact sensors – as well as increasing the contact surface. As an estimator attempts to find an accurate state configuration given sensor data, we stressed our estimator with these kinds of contact phenomena during dynamic motion.

A good estimator would have good sensor accuracy while also avoiding drifting of the robot. Accuracy is examined with Leave-One-Out tests that regenerate specific sensor values when those sensors are left out of the estimation process.

We validated our drifting tests by using the Phasespace markers as ground truth (and not in the estimator) after appropriately filtering out poor marker readings as not all markers were visible all the time. We attempt to validate two things. First, our estimator should

not have a high root mean squared error between its predicted Phasespace markers and the actual measured marker data, where the error at time k is the average Euclidean distance for the set of used markers.

$$RMSE_k = \frac{1}{N_{markers}} \sum_{markers} \|z_k - \hat{z}_k\|$$

To show our estimator also works over longer periods of time as well, we perform the same RMSE test with our process noise model and including the Phasespace markers in the estimator. Without the markers in the estimator, any estimator would be likely to drift over time.

Where here the z_k terms are only the marker measurements instead of the full sensor vector. We calculate this $RMSE$ for different configurations of model noise to show the combined effects. Secondly, we attempt to validate our model noise parameters by scaling its values and calculating an average $RMSE$ for the entire trajectory to show how changing these model noise terms affects the drift amount.

As we did not utilize the external position tracking of the Phasespace markers in our estimator, we initialized the estimator with our simulated models data after the simulation was configured with the same initial control vector of the actual robot. This meant that the estimator initialization began close to the actual true starting position – we assume the initialization of the robot hardware begins near the origin of the simulator's coordinate system in a robot centric fashion. Without any root position and orientation tracking system, divergence would be otherwise inevitable as the root of the robot is not directly observable.

3.6 RESULTS

Our estimation technique utilizing modeling noise successfully tracks our robot through dynamical motions with numerous contacts. Importantly, it does so while keeping drift low even while walking across a flat surface without external position sensing used for the estimate. We had implemented a basic EKF for comparison, but the filter diverged almost instantly

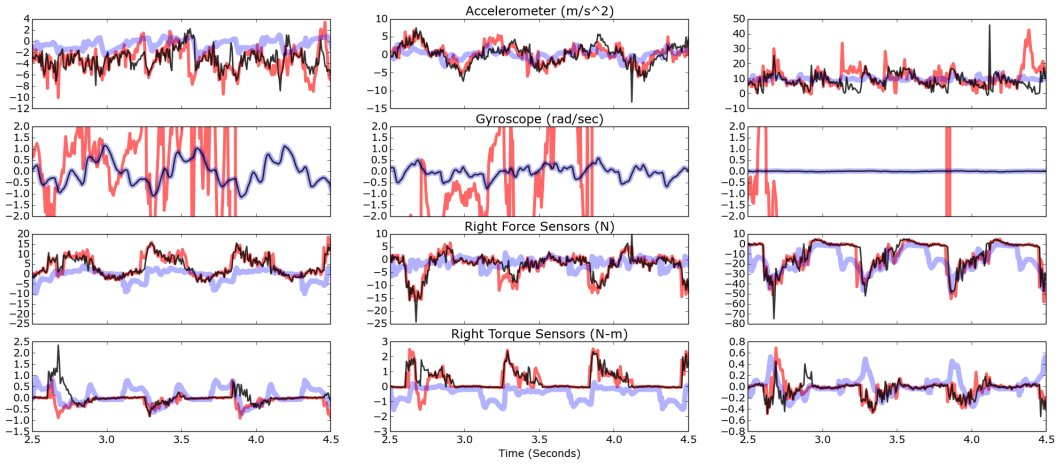


Figure 3.4: We show the results of a Leave-One-Out (LOO) test where the walking robot was estimating its state without the use of the indicated sensor, and we plot the prediction for the missing sensor. The thick blue line is the recorded sensor value from the hardware, while the red line is the value predicted when the sensor was disabled during estimation. For reference, the black line is the predicted sensor value with full sensors. This LOO test shows a correspondence between the sensor weighting and how closely it is matched.

due to heavy non-linearities from contacts. Thus, for brevity we only include data from our main UKF in this paper.

Figure 3.2 shows the effects of different configurations of model noise; the best performing configuration was with time, mass, and control noise all utilized for both the walking trajectory and fallen position. This intuitively makes sense as allowing for more noise in the estimator gives it more possible states to correct for. The plot shows one particular walking sequence of eight steps over five seconds with a drift of about 4 cm; the total distance travelled was 35.2 cm. Additionally, some estimator configurations with less noise would actually diverge. This is especially apparent in the configuration without contributing noise: the robot falls over in a configuration the Kalman update cannot correct for, leading to an over-correction and blow-up. However, it is important to note that the estimator does perform very well when

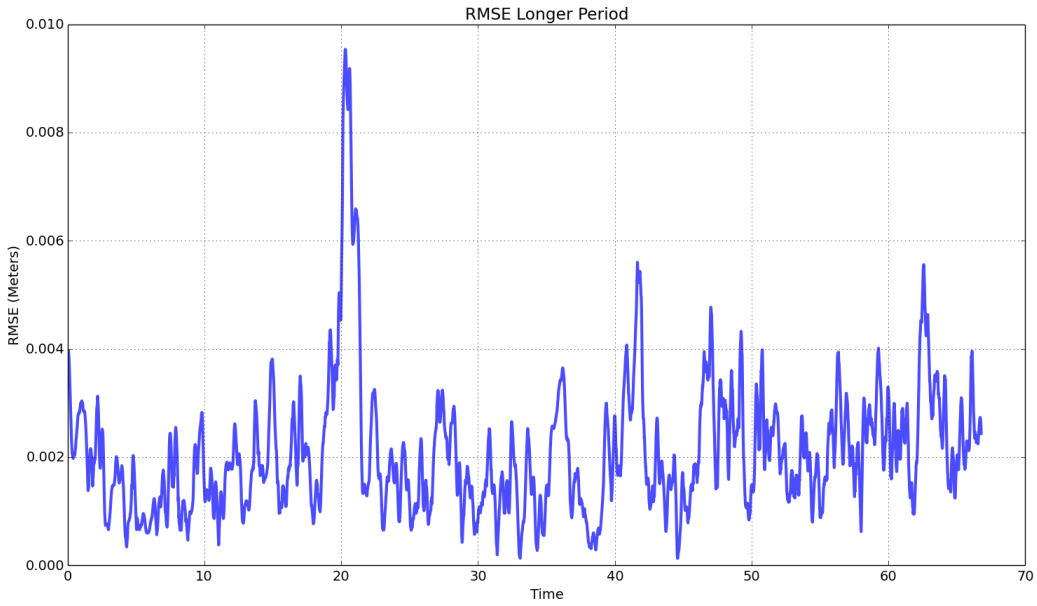


Figure 3.5: As drift is likely without high fidelity positional sensors, such as LIDAR, we include this example of our UKF tracking over a minute including the Phasespace sensors in the estimator. RMS error of this size can most likely be attributed to Phasespace calibration and system identification errors.

given all sources of noise. In a fallen position, there are many contacts occurring, with even more dynamic motion and contacts coming from the walking sequence. It is a great example of the estimator's strong ability to deal with multiple contacts.

Figure 3.4 shows the estimator's attempts to predict missing sensor values. The ability to predict these are necessary to demonstrate the physical consistency of the estimated state. This shows which sensors are critically necessary for accurate estimation. For example, the gyroscope is poorly estimated when left out causing large errors, but predicted almost exactly when provided to the estimator. This is because the gyroscope is a low variance sensor, and can thus be trusted more. Further insights are to be had when looking at the F/T sensors. The matching is poor, but not off base when the sensor is left out, especially along the X, Y

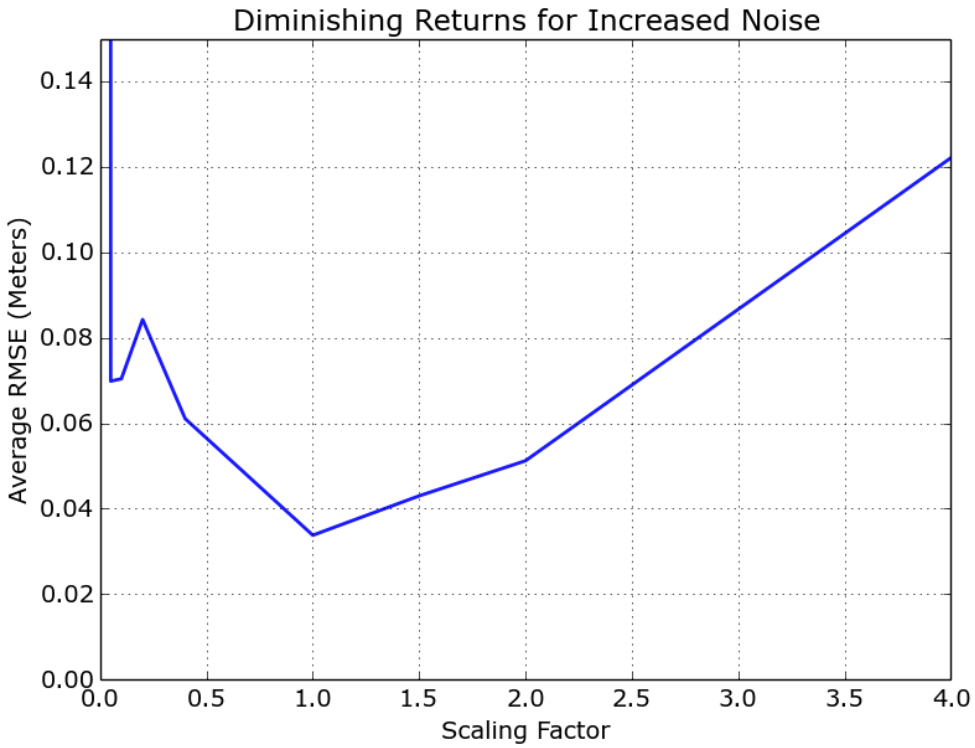


Figure 3.6: Here we show how our noise parameters contribute to estimator performance. Too little or too much added noise from time, control, and mass can cause a reduction in performance – in this case more drift. The RMSE values are averaged along the time of the trajectory to give us a measure of the mean deviation for the entire run.

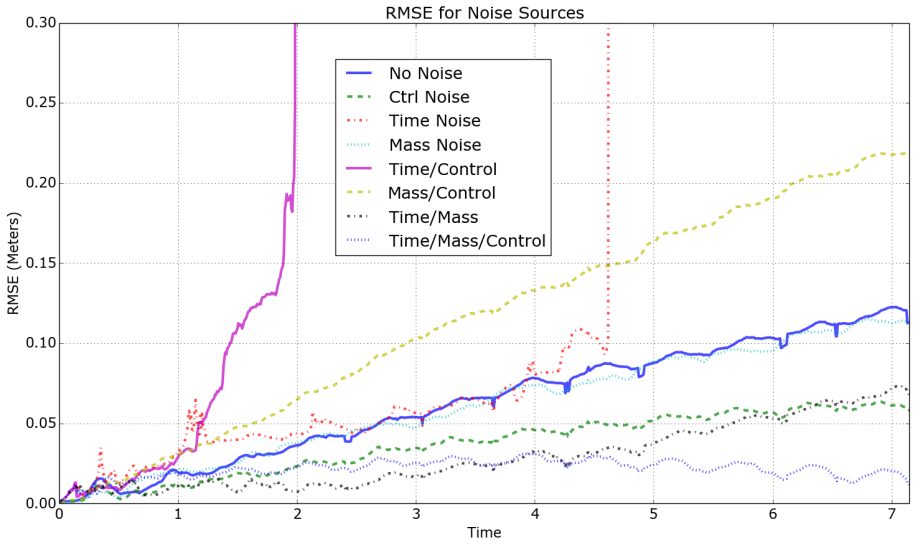


Figure 3.7: RMSE plot of when the robot was started in a fallen position. The walking sequence was still applied to induce dynamic motion and additional contacts. The effects of our estimator are not as apparent in this plot, although the configuration with all sources of noise still performs the best. This may be due to the fact that the robot does not traverse any distance, thus creating less chance for drift. It should be noted that some configurations of noise lead to blow-up in this fallen position; the extra contacts of the robot laying on the ground may have allowed estimator to go unstable from the noise inducing increased constraint violations.

Table 3.4: Avg. RMSE with Missing Sensors. We find that our estimator improves from using all available sensors (still not including the Phasespace markers).

Sensor	Avg. RMSE (Meters)
All	0.0407
No Accl	0.0493
No Gyro	7.76e+14
No Force	0.0434
No Torque	0.0809
No J.Pos	2.86e+9
No J.Vel	0.4215

axes. The Z axis values match more closely as they are more critical to drift-free forward walking and standing; these terms match more closely. The discrepancy of the X, Y axes can be partially explained by the sensitivity of modelling errors regarding these particular sensors. Small errors in contact geometry, hardness, or frictional coefficients may greatly affect these predicted values. By performing the Leave-One-Out test on all sensors, we gain insight on to how well they can be trusted, and thus how large the variance should be in the estimator. We also show how the noise parameters used by the estimator can also hurt the performance in figure 3.6 and table 3.4. The plot shows how scaling the sources of noise can actually increase RMSE values when too small or too large. Too little noise restricts the estimator's possible states, preventing it from making accurate corrections. Too much noise will throw off the estimator, causing it to diverge. The table calculates the average RMSE when specific sensors are not used in the estimation; there is divergence when the joint position and gyroscope sensors are not used.

3.7 Conclusions

We demonstrate our full body modified UKF estimator on a 26 DOF humanoid robotic in dynamic tasks with contacts. We are able to achieve low drift and accurate state and sensor reproduction with real-time performance characteristics. This was achieved without needing to 'soften' the rigid body contacts or splitting the estimate into two for root and joints, and without a reduced model. We were also able use only kinematic sensors at a reasonable sampling rate.

Further improvements could still be made. Though our attempt at handling estimate constraint violations (see section 3.2.1) resulted in inaccurate estimates, adjusting the UKF's sigma points to avoid un-realistic violations could improve the estimate. Instead of the line-search style approach, one idea is to project the sigma points from the previous timestep towards their new positions; this may allow some of the benefits of a MHE to occur.

Additionally, as our estimator works in real-time, future work should use this UKF estimator for dynamic feedback controls instead of the open-loop ZMP style walking we used here. More dynamic behavior may stress the estimator in its current configuration more; additional sources of modeling noise, including geometry, inertia orientations, and actuator parameters may be needed to improve its performance.

Chapter 4

GENERALIZING POLICY LEARNING

This chapter examines some fundamental assumptions about deep reinforcement learning approaches empirically. Namely, we attempt to address the question of what components contribute to successful deep reinforcement learning applications in high dimensional continuous control problems. Using the backdrop of the OpenAI Gym-v1 [27] suite of control benchmarks, this chapter shows that policies with simple linear and radial basis function parameterizations can be trained to solve a variety of widely studied continuous control tasks. The performance of these trained policies are competitive with state of the art results, obtained with more elaborate parameterizations such as fully connected neural networks. Recent works test their algorithms either exclusively or primarily on these tasks [126, 79, 52], and success on these tasks have been regarded as demonstrating a 'proof of concept'. Furthermore, the standard training and testing scenarios for these tasks are shown to be very limited and prone to over-fitting, thus giving rise to only trajectory-centric policies. Training with a diverse initial state distribution induces more global policies with better generalization.

Our contributions: Our results and their implications are highlighted below with more elaborate discussions in Section 3.5:

1. The success of recent RL efforts to produce rich motor behaviors have largely been attributed to the use of multi-layer neural network architectures. This work is among the first to carefully analyze the role of representation, and our results indicate that very simple policies including linear and RBF parameterizations are able to achieve state of the art results on widely studied tasks. Furthermore, such policies, particularly the linear ones, can be trained significantly faster (almost 20x) due to orders of magnitude fewer

parameters. This indicates that even for tasks with complex dynamics, there could exist relatively simple policies. This opens the door for studying a wide range of representations in addition to deep neural networks and understand trade-offs including computational time, theoretical justification, robustness, sample complexity etc.

2. We study these issues not only with regards to the performance metric at hand but we also take the further step in examining them in the context of robustness. Our results indicate that, with conventional training methods, the agent is able to successfully learn a limit cycle for walking, but cannot recover from any perturbations that are delivered to it. For transferring the success of RL to robotics, such brittleness is highly undesirable.

3. Finally, we directly attempt to learn more robust policies through using more diverse training conditions, which favor such policies. This is similar in spirit to the model ensemble approaches [87, 118] and domain randomization approaches [124, 148], which have successfully demonstrated improved robustness and simulation to real world transfer. Under these new and more diverse training scenarios, we again find that there is no compelling evidence to favor the use of multi-layer architectures, at least for the benchmark tasks. On a side note, we also provide interactive testing of learned policies, which we believe is both novel and which sheds light on the robustness of trained policies.

Overall, we note that this work does not attempt to provide a definitive answer in terms of the ideal architecture choice for control. Rather, the results in this work suggest that the current set of benchmark tasks are insufficient to provide insights to this question. We further note that as the research field progresses, it is imperative to revisit these questions to make well calibrated progress.

4.1 Problem Formulation and Methods

We consider Markov Decision Processes (MDPs) in the average reward setting, which is defined using the tuple: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0\}$. $\mathcal{S} \subseteq \mathbb{R}^n$, $\mathcal{A} \subseteq \mathbb{R}^m$, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are a (continuous) set of states, set of actions, and reward function respectively, and have

the usual meaning. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the stochastic transition function and ρ_0 is the probability distribution over initial states. We wish to solve for a stochastic policy of the form $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$, which optimizes the objective function:

$$\eta(\pi) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_{\pi, \mathcal{M}} \left[\sum_{t=1}^{T} r_t \right]. \tag{4.1}$$

Since we use simulations with finite length rollouts to estimate the objective and gradient, we approximate $\eta(\pi)$ using a finite T. In this finite horizon rollout setting, we define the value, Q, and advantage functions as follows:

$$V^{\pi}(s,t) = \mathbb{E}_{\pi, \mathcal{M}} \left[\sum_{t'=t}^{T} r_{t'} \right]$$

$$Q^{\pi}(s,a,t) = \mathbb{E}_{\mathcal{M}} \Big[\mathcal{R}(s,a) \Big] + \mathbb{E}_{s' \sim \mathcal{T}(s,a)} \Big[V^{\pi}(s', t+1) \Big]$$

$$A^{\pi}(s,a,t) = Q^{\pi}(s,a,t) - V^{\pi}(s,t)$$

Note that even though the value functions are time-varying, we still optimize for a stationary policy. We consider parametrized policies π_θ, and hence wish to optimize for the parameters (θ). Thus, we overload notation and use $\eta(\pi)$ and $\eta(\theta)$ interchangeably.

4.1.1 Algorithm

Ideally, a controlled scientific study would seek to isolate the challenges related to architecture, task design, and training methods for separate study. In practice, this is not entirely feasible as the results are partly coupled with the training methods through the interaction of the reward function and environment dynamics changing the optimization landscape in complex ways.

Here, we utilize a straightforward natural policy gradient method for training. The work in [38] suggests that this method is competitive with most state of the art methods. We now discuss the training procedure.

Using the likelihood ratio approach and Markov property of the problem, the sample based estimate of the policy gradient is derived to be [156]:

$$\nabla_\theta \hat{\eta}(\theta) = g = \frac{1}{T} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}^\pi(s_t, a_t, t) \tag{4.2}$$

Gradient ascent using this "vanilla" gradient is sub-optimal since it is not the steepest ascent direction in the metric of the parameter space [2, 63]. The steepest ascent direction is obtained by solving the following local optimization problem around iterate θ_k:

$$\underset{\theta}{\text{maximize}} \quad g^T(\theta - \theta_k) \qquad \text{subject to} \quad (\theta - \theta_k)^T F_{\theta_k} (\theta - \theta_k) \leq \delta, \tag{4.3}$$

where F_{θ_k} is the Fisher Information Metric at the current iterate θ_k. We estimate F_{θ_k} as

$$\hat{F}_{\theta_k} = \frac{1}{T} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^T, \tag{4.4}$$

as originally suggested by Kakade [63]. This yields the steepest ascent direction to be $\hat{F}_{\theta_k}^{-1} g$ and corresponding update rule: $\theta_{k+1} = \theta_k + \alpha \hat{F}_{\theta_k}^{-1} g$. Here α is the step-size or learning rate parameter. Empirically, we observed that choosing a fixed value for α or an appropriate schedule is difficult [106]. Thus, we use the normalized gradient ascent procedure, where the normalization is under the Fisher metric. This procedure can be viewed as picking a normalized step size δ as opposed to α, and solving the optimization problem in (3). This results in the following update rule:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T \hat{F}_{\theta_k}^{-1} g}} \; \hat{F}_{\theta_k}^{-1} g. \tag{4.5}$$

A dimensional analysis of these quantities reveal that α has the unit of return^{-1} whereas δ is dimensionless. Though units of α are consistent with a general optimization setting where step-size has units of objective^{-1}, in these problems, picking a good α that is consistent with the scales of the reward was difficult. On the other hand, a constant normalized step size was numerically more stable and easier to tune: for all the results reported in this paper, the same $\delta = 0.05$ was used. When more than one trajectory rollout is used per update, the above estimators can be used with an additional averaging over the trajectories.

Algorithm 1 Policy Search with Natural Gradient

1: Initialize policy parameters to θ_0
2: **for** $k = 1$ **to** K **do**
3: Collect trajectories $\{\tau^{(1)}, \ldots \tau^{(N)}\}$ by rolling out the stochastic policy $\pi(\cdot; \theta_k)$.
4: Compute $\nabla_\theta \log \pi(a_t|s_t; \theta_k)$ for each (s, a) pair along trajectories sampled in iteration k.
5: Compute advantages A_k^π based on trajectories in iteration k and approximate value function V_{k-1}^π.
6: Compute policy gradient according to (2).
7: Compute the Fisher matrix (4) and perform gradient ascent (5).
8: Update parameters of value function in order to approximate $V_k^\pi(s_t^{(n)}) \approx R(s_t^{(n)})$, where $R(s_t^{(n)})$ is the empirical return computed as $R(s_t^{(n)}) = \sum_{t'=t}^{T} \gamma^{(t'-t)} r_t^{(n)}$. Here n indexes over the trajectories.
9: **end for**

For estimating the advantage function, we use the GAE procedure [126]. This requires learning a function that approximates V_k^π, which is used to compute A_k^π along trajectories for the update in (5). GAE helps with variance reduction at the cost of introducing bias, and requires tuning hyperparameters like a discount factor and an exponential averaging term. Good heuristics for these parameters have been suggested in prior work. The same batch of trajectories cannot be used for both fitting the value function baseline, and also to estimate g using (2), since it will lead to overfitting and a biased estimate. Thus, we use the trajectories from iteration $k-1$ to fit the value function, essentially approximating V_{k-1}^π, and use trajectories from iteration k for computing A_k^π and g. Similar procedures have been adopted in prior work [38].

4.1.2 Policy Architecture

Linear policy: We first consider a linear policy that directly maps from the observations to the motor torques. We use the same observations as used in prior work which includes joint positions, joint velocities, and for some tasks, information related to contacts. Thus, the policy mapping is $a_t \sim \mathcal{N}(Ws_t + b, \sigma)$, and the goal is to learn W, b, and σ. For most of these tasks, the observations correspond to the state of the problem (in relative coordinates). Thus, we use the term states and observations interchangeably. In general, the policy is defined with observations as the input, and hence is trying to solve a POMDP.

RBF policy: Secondly, we consider a parameterization that enriches the representational capacity using random Fourier features of the observations. Since these features approximate the RKHS features under an RBF Kernel [115], we call this policy parametrization the RBF policy. The features are constructed as:

$$y_t^{(i)} = \sin\left(\frac{\sum_j P_{ij} s_t^{(j)}}{\nu} + \phi^{(i)}\right), \tag{4.6}$$

where each element P_{ij} is drawn from $\mathcal{N}(0, 1)$, ν is a bandwidth parameter chosen approximately as the average pairwise distances between different observation vectors, and ϕ is a random phase shift drawn from $U[-\pi, \pi)$. Thus the policy is $a_t \sim \mathcal{N}(Wy_t + b, \sigma)$, where W, b, and σ are trainable parameters. This architecture can also be interpreted as a two layer neural network: the bottom layer is clamped with random weights, a sinusoidal activation function is used, and the top layer is learned. The principal purpose for this representation is to slightly enhance the capacity of a linear policy, and the choice of activation function is not very significant.

4.2 Results on OpenAI gym-v1 benchmarks

As indicated before, we train linear and RBF policies with the natural policy gradient on the popular OpenAI gym-v1 benchmark tasks simulated in MuJoCo [150]. The tasks primarily

consist of learning locomotion gaits for simulated robots ranging from a swimmer to a 3D humanoid (23 dof).

Figure 4.1 presents the learning curves along with the performance levels reported in prior work using TRPO and fully connected neural network policies. Table 1 also summarizes the final scores, where "stoc" refers to the stochastic policy with actions sampled as $a_t \sim \pi_\theta(s_t)$, while "mean" refers to using mean of the Gaussian policy, with actions computed as $a_t = \mathbb{E}[\pi_\theta(s_t)]$. We see that the linear policy is competitive on most tasks, while the RBF policy can outperform previous results on five of the six considered tasks. Though we were able to train neural network policies that match the results reported in literature, we have used publicly available prior results for an objective comparison. Visualizations of the trained linear and RBF policies are presented in the supplementary video. Given the simplicity of these policies, it is surprising that they can produce such elaborate behaviors.

Table 4.2 presents the number of samples needed for the policy performance to reach a threshold value for reward. The threshold value is computed as 90% of the final score achieved by the stochastic linear policy. We visually verified that policies with these scores are proficient at the task, and hence the chosen values correspond to meaningful performance thresholds. We see that linear and RBF policies are able to learn faster on four of the six tasks.

All the simulated robots we considered are under-actuated, have contact discontinuities, and continuous action spaces making them challenging benchmarks. When adapted from model-based control [41, 141, 40] to reinforcement learning, however, the notion of "success" established was not appropriate. To shape the behavior, a very narrow initial state distribution and termination conditions are used in the benchmarks. As a consequence, the learned policies become highly trajectory-centric – i.e. they are good only where they tend to visit during training, which is a very narrow region. For example, the walker can walk very well when initialized upright and close to the walking limit cycle. Even small perturbations, as shown in the supplementary video, alters the visitation distribution and dramatically degrades the policy performance. This makes the agent fall down at which point it is unable to get up.

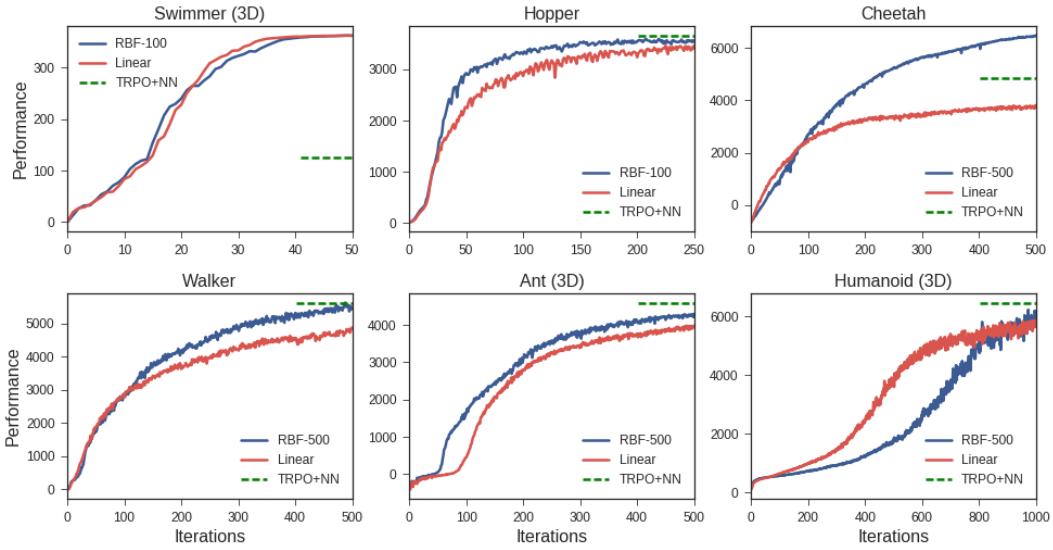


Figure 4.1: Learning curves for the Linear and RBF policy architectures. The green line corresponding to the reward achieved by neural network policies on the OpenAI Gym website, as of 02/24/2017 (trained with TRPO). It is observed that for all the tasks, linear and RBF parameterizations are competitive with state of the art results. The learning curves depicted are for the stochastic policies, where the actions are sampled as $a_t \sim \pi_\theta(s_t)$. The learning curves have been averaged across three runs with different random seeds.

Similarly, the swimmer is unable to turn when its heading direction is altered. For control applications, this is undesirable. In the real world, there will always be perturbations – stochasticity in the environment, modeling errors, or wear and tear. Thus, the specific task design and notion of success used for the simulated characters are not adequate. However, the simulated robots themselves are rather complex and harder tasks could be designed with them, as partly illustrated in Section 3.4.

Task	**Linear**		**RBF**		**NN**
	stoc	mean	stoc	mean	TRPO
Swimmer	362	**366**	361	365	131
Hopper	3466	3651	3590	**3810**	3668
Cheetah	3810	4149	6477	**6620**	4800
Walker	4881	5234	5631	**5867**	5594
Ant	3980	4607	4297	4816	**5007**
Humanoid	5873	6440	6237	**6849**	6482

Table 4.1: Final performances of the policies

Task	**Th.**	**Linear**	**RBF**	**TRPO+NN**
Swimmer	325	**1450**	1550	N-A
Hopper	3120	13920	**8640**	10000
Cheetah	3430	11250	6000	**4250**
Walker	4390	36840	25680	**14250**
Ant	3580	39240	**30000**	73500
Humanoid	5280	**79800**	96720	87000

Table 4.2: Number of episodes to achieve threshold

4.3 Modified Tasks and Results

Using the same set of simulated robot characters outlined in Section 3.3, we designed new tasks with two goals in mind: (a) to push the representational capabilities and test the limits of simple policies; (b) to enable training of “global" policies that are robust to perturbations and work from a diverse set of states. To this end, we make the following broad changes

1. Wider initial state distribution to force generalization. For example, in the walker task, some fraction of trajectories have the walker initialized prone on the ground. This forces the agent to simultaneously learn a get-up skill and a walk skill, and not forget them as the learning progresses. Similarly, the heading angle for the swimmer and ant are randomized, which encourages learning of a turn skill.

2. Reward shaping appropriate with the above changes to the initial state distribution. For example, when the modified swimmer starts with a randomized heading angle, we include a small reward for adjusting its heading towards the correct direction. In conjunction, we also remove all termination conditions used in the Gym-v1 benchmarks.

3. Changes to environment's physics parameters, such as mass and joint torque. If the agent has sufficient power, most tasks are easily solved. By reducing an agent's action ability and/or increasing its mass, the agent is more under-actuated. These changes also produce more realistic looking motion.

Combined, these modifications require that the learned policies not only make progress towards maximizing the reward, but also recover from adverse conditions and resist perturbations. An example of this is illustrated in Figure 4.2, where the hopper executes a get-up sequence before hopping to make forward progress. Furthermore, at test time, a user can interactively apply pushing and rotating perturbations to better understand the failure modes. We note that these interactive perturbations may not be the ultimate test for robustness, but

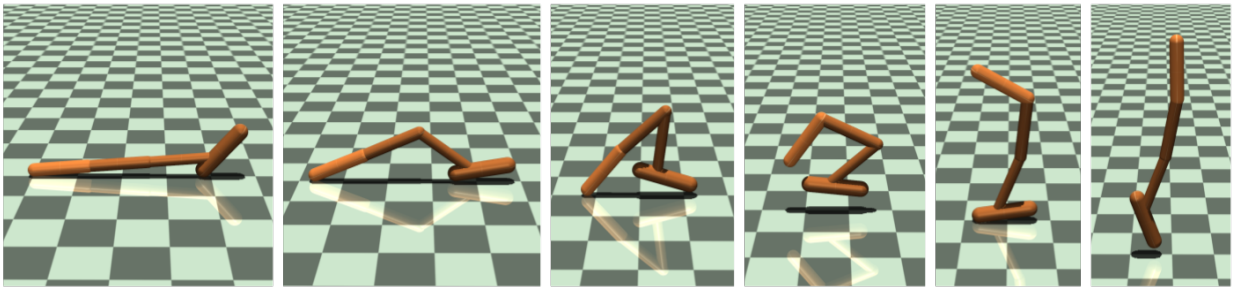


Figure 4.2: Hopper completes a get-up sequence before moving to its normal forward walking behavior. The getup sequence is learned along side the forward hopping in the modified task setting.

a step towards this direction, and represents the interactive perturbations a robotic system might experience during runtime.

Representational capacity The supplementary video demonstrates the trained policies. We concentrate on the results of the walker task in the main paper. Figure 4.3 studies the performance as we vary the representational capacity. Increasing the Fourier features allows for more expressive policies and consequently allow for achieving a higher score. The policy with 500 Fourier features performs the best, followed by the fully connected neural network. The linear policy also makes forward progress and can get up from the ground, but is unable to learn as efficient a walking gait.

Perturbation resistance Next, we test the robustness of our policies by perturbing the system with an external force. This external force represents an unforeseen change which the agent has to resist or overcome, thus enabling us to understand push and fall recoveries. Fall recoveries of the trained policies are demonstrated in the supplementary video. In these tasks, perturbations are not applied to the system during the training phase. Thus, the ability to generalize and resist perturbations come entirely out of the states visited by the agent during

Table 4.3: Modified Task Description

v_x is forward velocity; θ is heading angle; p_z is the height of torso; and a is the action.

Task	Description	Reward (des = desired value)
Swimmer (3D)	Agent swims in the desired direction Should recover (turn) if rotated around	$v_x - 0.1\|\theta - \theta^{\text{des}}\| - 0.0001\|\|a\|\|^2$
Hopper (2D)	Agent hops forward as fast as possible Should recover (get up) if pushed down	$v_x - 3\|\|p_z - p_z^{\text{des}}\|\|^2 - 0.1\|\|a\|\|^2$
Walker (2D)	Agent walks forward as fast as possible Should recover (get up) if pushed down	$v_x - 3\|\|p_z - p_z^{\text{des}}\|\|^2 - 0.1\|\|a\|\|^2$
Ant (3D)	Agent moves in the desired direction Should recover (turn) if rotated around	$v_x - 3\|\|p_z - p_z^{\text{des}}\|\|^2 - 0.01\|\|a\|\|^2$

training. Figure 4.5 indicates that the RBF policy is more robust, and also that diverse initializations are important to obtain the best results. This indicates that careful design of initial state distributions are crucial for generalization, and to enable the agent to learn a wide range of skills.

4.4 Summary and Discussion

The experiments in this paper were aimed at trying to understand the effects of (a) representation; (b) task modeling; and (c) optimization. We summarize the results with regard to each aforementioned factor and discuss their implications.

Representation The finding that linear and RBF policies can be trained to solve a variety of continuous control tasks is very surprising. Recently, a number of algorithms have been shown to successfully solve these tasks [125, 54, 79, 52], but all of these works use multi-layer neural networks. This suggests a widespread belief that expressive function approximators

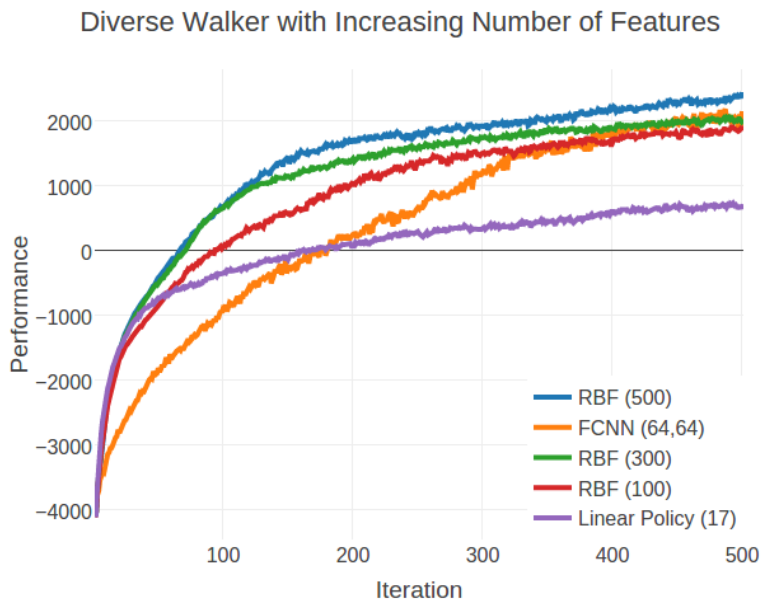


Figure 4.3: Learning curve on modified walker (diverse initialization) for different policy architectures. The curves are averaged over three runs with different random seeds.

are needed to capture intricate details necessary for movements like running. The results in this work conclusively demonstrates that this is not the case, at least for the limited set of popular testbeds. This raises an interesting question: what are the capability limits of shallow policy architectures? The linear policies were not exemplary in the "global" versions of the tasks, but it must be noted that they were not terrible either. The RBF policy using random Fourier features was able to successfully solve the modified tasks producing global policies, suggesting that we do not yet have a sense of its limits.

Modeling When using RL methods to solve practical problems, the world provides us with neither the initial state distribution nor the reward. Both of these must be designed by the researcher and must be treated as assumptions about the world or prescriptions about the required behavior. The quality of assumptions will invariably affect the quality of solutions, and thus care must be taken in this process. Here, we show that starting the system from

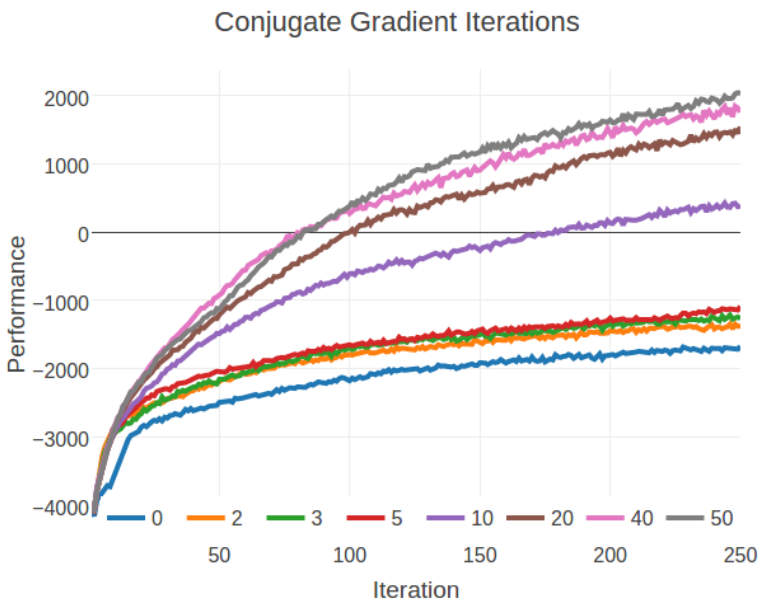


Figure 4.4: Learning curves when using different number of conjugate gradient iterations to compute $\hat{F}_{\theta_k}^{-1} g$ in equation 4.5. A policy with 300 Fourier features has been used to generate these results.

a narrow initial state distribution produces elaborate behaviors, but the trained policies are very brittle to perturbations. Using a more diverse state distribution, in these cases, is sufficient to train robust policies.

Optimization In line with the theme of simplicity, we first tried to use REINFORCE [156], which we found to be very sensitive to hyperparameter choices, especially step-size. There are a class of policy gradient methods which use pre-conditioning to help navigate the warped parameter space of probability distributions and for step size selection. Most variants of pre-conditioned policy gradient methods have been reported to achieve state of the art performance, all performing about the same [38]. We feel that the used natural policy gradient method is the most straightforward pre-conditioned method. To demonstrate that the pre-conditioning helps, Figure 4.4 depicts the learning curve for different number of CG

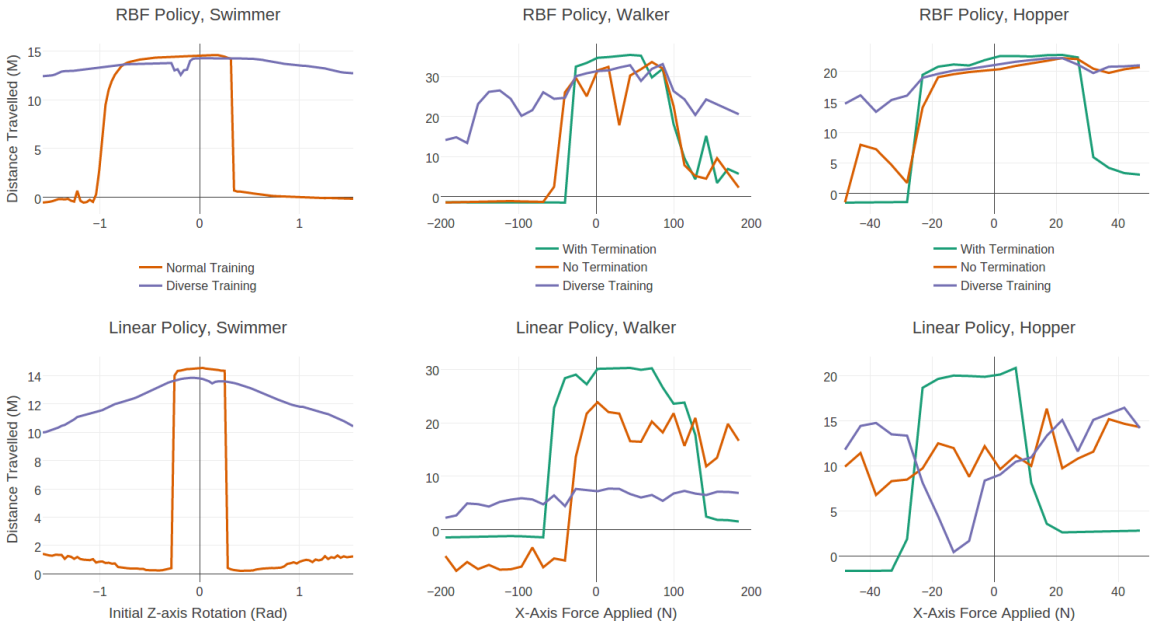


Figure 4.5: We test policy robustness by measuring distanced traveled in the swimmer, walker, and hopper tasks for three training configurations: (a) with termination conditions; (b) no termination, and peaked initial state distribution; and (c) with diverse initialization. Swimmer does not have a termination option, so we consider only two configurations. For the case of swimmer, the perturbation is changing the heading angle between $-\pi/2.0$ and $\pi/2.0$, and in the case of walker and hopper, an external force for 0.5 seconds along its axis of movement. All agents are initialized with the same positions and velocities.

iterations used to compute the update in (5). The curve corresponding to $CG = 0$ is the REINFORCE method. As can be seen, pre-conditioning helps with the learning process. However, there is a trade-off with computation, and hence using an intermediate number of CG steps like 20 could lead to best results in wall-clock sense for large scale problems.

We chose to compare with neural network policies trained with TRPO, since it has demonstrated impressive results and is closest to the algorithm used in this work. Are function approximators linear with respect to free parameters sufficient for other methods

is an interesting open question (in this sense, RBFs are linear but NNs are not). For a large class of methods based on dynamic programming (including Q-learning, SARSA, approximate policy and value iteration), linear function approximation has guaranteed convergence and error bounds, while non-linear function approximation is known to diverge in many cases [49, 130, 19, 11]. It may of course be possible to avoid divergence in specific applications, or at least slow it down long enough, for example via target networks or replay buffers. Nevertheless, guaranteed convergence has clear advantages. Similar to recent work using policy gradient methods, recent work using dynamic programming methods have adopted multi-layer networks without careful side-by-side comparisons to simpler architectures. Could a global quadratic approximation to the optimal value function (which is linear in the set of quadratic features) be sufficient to solve most of the continuous control tasks currently studied in reinforcement learning? Given that quadratic value functions correspond to linear policies, and good linear policies exist as shown here, this might make for interesting future work.

4.5 Conclusion

In this work, we demonstrated that very simple policy parameterizations can be used to solve many benchmark continuous control tasks. Furthermore, there is no significant loss in performance due to the use of such simple parameterizations. We also proposed global variants of many widely studied tasks, which requires the learned policies to be competent for a much larger set of states, and found that simple representations are sufficient in these cases as well. These empirical results along with Occam's razor suggests that complex policy architectures should not be a default choice unless side-by-side comparisons with simpler alternatives suggest otherwise. Such comparisons are unfortunately not widely pursued. The results presented in this work directly highlight the need for simplicity and generalization in reinforcement learning. The alternative interpretation is that the capabilities of these methods are not being pushed to their fullest, and the perceived challenges that these algorithms may have is not due to the representation, but rather the quality of the data

samples they experience. We hope that this work would encourage future work analyzing various architectures and associated trade-offs like computation time, robustness, and sample complexity.

Chapter 5

TRANSFER OF LEARNED POLICIES FROM SIMULATION TO PHYSICAL SYSTEM

Reinforcement learning has emerged as a promising methodology for training robot controllers. However, most results have been limited to simulation due to the need for a large number of samples and the lack of automated-yet-safe data collection methods. Model-based reinforcement learning methods provide an avenue to circumvent these challenges, but the traditional concern has been the mismatch between the simulator and the real world. In this chapter, we show that control policies learned in simulation can successfully transfer to a physical system, composed of three Phantom robots pushing an object to various desired target positions. We use a modified form of the natural policy gradient algorithm for learning, applied to a carefully identified simulation model. The resulting policies, trained entirely in simulation, work well on the physical system without additional training. In addition, we show that training with an ensemble of models makes the learned policies more robust to modeling errors, thus compensating for difficulties in system identification.

Non-prehensile object manipulation remains a challenging control problem in robotics. In this work, we focus on a particularly challenging system using three Phantom robots as fingers. These are haptic robots that are torque-controlled and have higher bandwidth than the fingers of existing robotic hands. In terms of speed and compliance (but not strength), they are close to the capabilities of the human hand. This makes them harder to control, especially in non-prehensile manipulation tasks where the specifics of each contact event and the balance of contact forces exerted on the object are very important and need to be considered by the controller in some form.

Here we develop a solution using Reinforcement Learning (RL). We use the MuJoCo

physics simulator [150] as the modeling platform and fit the parameters of the simulation model to match the real system as closely as possible. For policy learning with the model, we use a normalized natural policy gradient (NPG) method [63, 105, 121]. While Reinforcement Learning (RL) methods such as NPG are in principle model-free, in practice they require large amounts of data. In the absence of an automatic way to generate safe exploration controllers, learning is largely possible only in simulation. Indeed the vast majority of recent results in continuous RL have been obtained in simulation. These studies often propose to extend the corresponding methods to physical systems in future work, but the scarcity of such results indicates that 'sim-to-real' transfer is harder than it seems. The few successful applications to real robots have been in tasks involving position or velocity control that avoid some difficulties of torque-controlled platforms.

To obtain an accurate simulation model, we fit the parameters of the simulation model using a physically consistent system identification procedure [68] developed specifically for identification in contact rich settings such as the one we study. While true model-free RL may one day become feasible, we believe leveraging the capabilities of a physics simulator will always help speed up the learning process.

5.1 Hardware and Physics Simulation

5.1.1 System Overview

As with any controller developed in simulation, performance on the real system is likely to degrade due to modeling errors. To assess, as well as mitigate, the effects of these errors, we compare learning with respect to three different models: (i) the nominal model obtained from system identification; (ii) a modified model where we intentionally mis-specify some model parameters; (iii) an ensemble of models where the mean is wrong but the variance is large enough to include the nominal model. The purpose of (ii) is to simulate a scenario where system identification has not been performed well, and we wish to study the performance degradation. The goal with ensemble approaches [87, 118, 104] is to study if including a

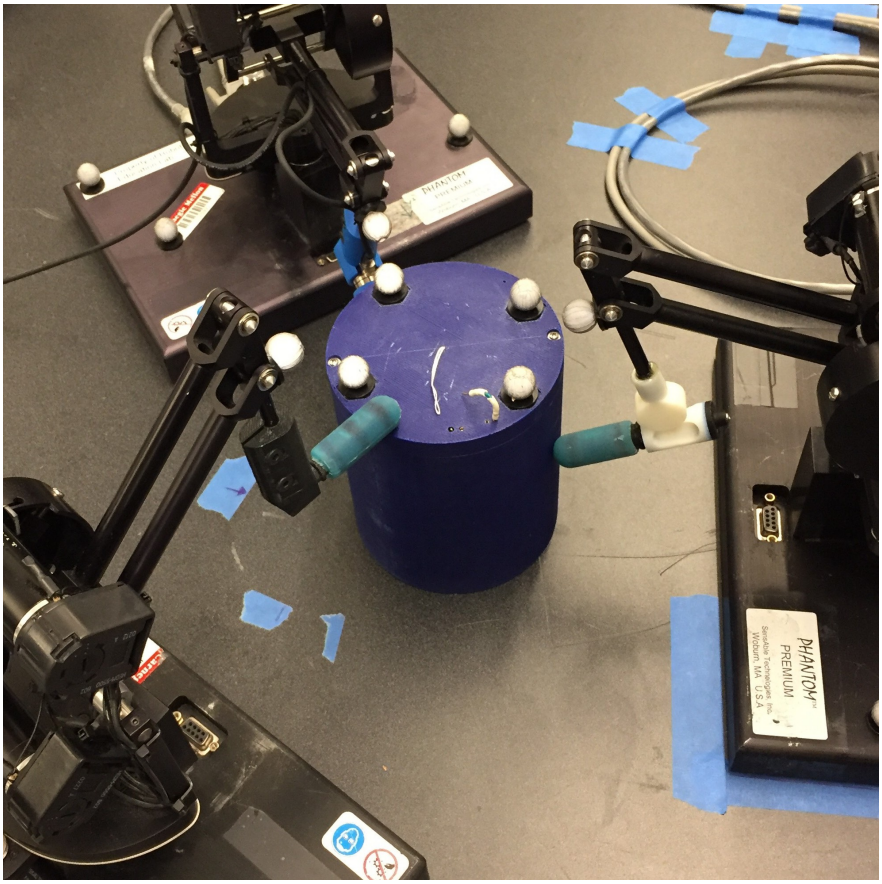


Figure 5.1: Phantom Manipulation Platform.

distribution over models can compensate for inaccuracies in system identification during control tasks. We find that (i) achieves the best performance as expected, and (iii) is robust but suffers a degradation in performance.

5.2 Problem Formulation

We model the control problem as a Markov Decision Process (MDP) in the episodic average reward setting, which is defined using the tuple: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0, T\}$. $\mathcal{S} \subseteq \mathbb{R}^n$, $\mathcal{A} \subseteq \mathbb{R}^m$, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are (continuous) set of states, set of actions, and the reward function respectively. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the stochastic transition function; ρ_0 is the probability distribution over initial states; and T is the maximum episode length. We wish to solve for a

stochastic policy of the form $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which optimizes the average reward accumulated over the episode. Formally, the performance of a policy is evaluated according to:

$$\eta(\pi) = \frac{1}{T}\, \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t=1}^{T} r_t\right]. \tag{5.1}$$

In this finite horizon rollout setting, we define the value, Q, and advantage functions as follows:

$$V^{\pi}(s,t) = \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t'=t}^{T} r_{t'}\right]$$

$$Q^{\pi}(s,a,t) = \mathbb{E}_{\mathcal{M}}\Big[\mathcal{R}(s,a)\Big] + \mathbb{E}_{s'\sim\mathcal{T}(s,a)}\Big[V^{\pi}(s',t+1)\Big]$$

$$A^{\pi}(s,a,t) = Q^{\pi}(s,a,t) - V^{\pi}(s,t)$$

We consider parametrized policies π_θ, and hence wish to optimize for the parameters (θ). In this work, we represent π_θ as a multivariate Gaussian with diagonal covariance. In our experiments, we use an affine policy as our function approximator, visualized in figure 5.3.

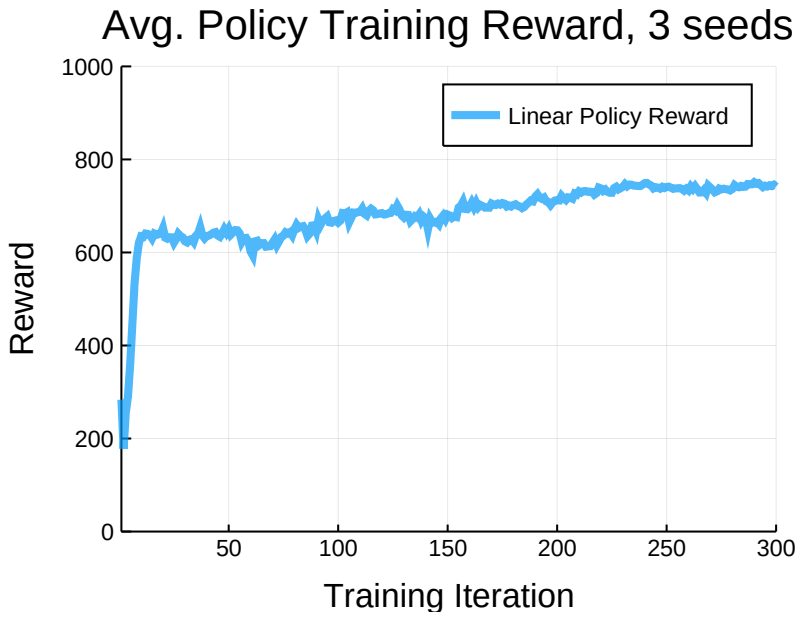


Figure 5.2: Learning curve of our linear/affine policy. We show here the curve for the policy trained with the correct mass as a representative curve.

Algorithm 2 Distributed Natural Policy Gradient

Initialize policy parameters to θ_0
for $k = 1$ **to** K **do**
  Distribute Policy and Value function parameters.
  for $w = 1$ **to** $N_{workers}$ **do**
    Collect trajectories $\{\tau^{(1)}, \dots \tau^{(N)}\}$ by rolling out the stochastic policy $\pi(\cdot; \theta_k)$.
    Compute $\nabla_\theta \log \pi(a_t|s_t; \theta_k)$ for each (s, a) pair along trajectories sampled in iteration k.
    Compute advantages A_k^π based on trajectories in iteration k and approximate value function V_{k-1}^π.
    Compute policy gradient according to eq. (2).
    Compute the Fisher matrix (4).
    Return Fisher Matrix, gradient, and value function parameters to central server.
  end for
  Average Fisher Matrix gradient, and perform gradient ascent (5)
  Update parameters of value function.
end for

5.3 Method

5.3.1 Natural Policy Gradient

Policy gradient algorithms are a class of reinforcement learning methods where the parameters of the policy are directly optimized typically using gradient based methods. Using the score function gradient estimator, the sample based estimate of the policy gradient can be derived to be: [156]:

$$\hat{g} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \hat{A}^\pi(s_t^i, a_t^i, t) \tag{5.2}$$

A straightforward gradient ascent using the above gradient estimate is the REINFORCE algorithm [156]. Gradient ascent with this direction is sub-optimal since it is not the steepest

ascent direction in the metric of the parameter space [2]. Consequently, a local search approach that moves along the steepest ascent direction was proposed by Kakade [63] called the natural policy gradient (NPG). This has been expanded upon in subsequent works [105, 121, 106, 125, 127], and forms a critical component in state of the art RL algorithms. Natural policy gradient is obtained by solving the following local optimization problem around iterate θ_k:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & g^T(\theta - \theta_k) \\ \text{subject to} \quad & (\theta - \theta_k)^T F_{\theta_k}(\theta - \theta_k) \leq \delta, \end{aligned} \tag{5.3}$$

where F_{θ_k} is the Fisher Information Metric at the current iterate θ_k. We apply a normalized gradient ascent procedure, which has been shown to further stabilize the training process [106, 125, 121]. This results in the following update rule:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T F_{\theta_k}^{-1} g}}\, F_{\theta_k}^{-1} g. \tag{5.4}$$

The version of natural policy gradient outlined above was chosen for simplicity and ease of implementation. The natural gradient performs covariant updates by rescaling the parameter updates according to curvature information present in the Fisher matrix, thus behaving almost like a second order optimization method. Furthermore, due to the normalized gradient procedure, the gradient information is insensitive to linear rescaling of the reward function, improving training stability. For estimating the advantage function, we use the GAE procedure [126] and use a quadratic function approximator with all the terms in s for the baseline.

5.3.2 Distributed Processing

As the natural policy gradient algorithm is an on-policy method, all data is collected from the current policy. However, the NPG algorithm allows for the rollouts and most computation to be performed independently as only the gradient and the Fisher matrix need to be synced. Independent processes can compute the gradient and Fisher matrix, with a centralized server

averaging these values and performing the matrix inversion and gradient step as in equation (4). The new policy is then shared with each worker. The total size of messages passed is proportional to the size of the Fisher Matrix used for the policy, and linear in the number of worker nodes. Policies with many parameters may experience message passing overhead, but the trade-off is that each worker can perform as many rollouts during sample collection without changing the message size, encouraging more data gathering (which large policies require).

A summary of the distributed algorithm we used is show in Algorithm 2.

We implemented our distributed NPG algorithm and interfaced with the MuJoCo simulator with the Julia programming language [20]. The built-in multi-processing and multi-node capabilities of Julia facilitated this distributed algorithm's performance; we are able to train a linear policy on this task in less than 3 minutes on a 4 node cluster with Intel i7-3930k processors.

We use our Phantom Manipulation Platform as our hardware testbed. It consists of three Phantom Haptic Devices, each acting as a robotic finger. Each haptic device is a 3-DOF cable driven system shown in figure 5.1. The actuation is done with Maxon motors (Model RE 25 #118743), three per Phantom. Despite the low gear reduction ratio, they are able to achieve 8.5N instantaneous force and 0.6N continuous force at the middle of their range of motion, with very low friction for the entire range of motion.

The three robots are coupled together to act as one manipulator. Each robot's end effector was equipped with a silicon covered fingertip to enable friction and reliable grasping of objects. The softness of the silicon coating was an additional challenge in both contact modeling and robust policy learning. For this work, we had the robots manipulating a 3D printed cylinder with a height of 14cm and diameter of 11cm, and a mass of 0.34kg.

The soft contacts, combined with the direct torque control and high power-to-weight ratio–leading to high acceleration–make this platform particularly difficult to control. Systems with more mass and natural damping in their joints naturally move more slowly and smoothly; this is not the case here. Being able to operate in this space, however, allows for the potential

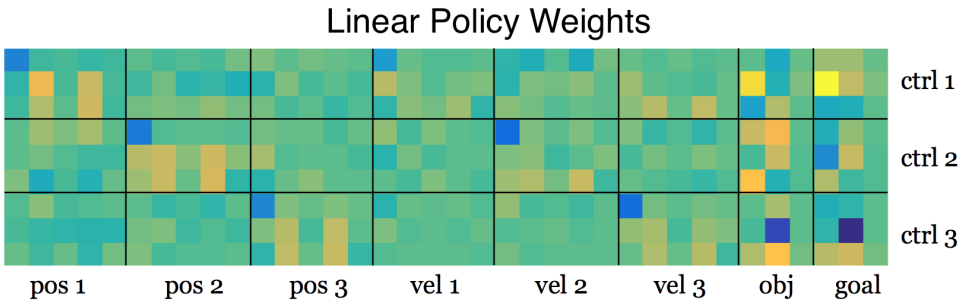


Figure 5.3: We render the policy parameters. A distinct pattern can be seen in the three negative weights in the top left of the pos and vel blocks. These correspond to the control outputs for the each of the three Phantom's turret joint; as each robot is sensitive, the policy outputs a negative gain to avoid large control forces. Additionally, we can see that the first and second Phantom contributes primarily the object's X location, while the third Phantom handles the Y location. This linear policy can be likened to a hand crafted feedback policy, but the numerous non-zero values would be unintuitive to select for a human. Presumably they can be used to contribute to the controllers robustness as learned through this method.

for high performance, dynamic manipulation, and the benefits that come with torque based control. However, this requires that we operate our robot controller at 2kHz to successfully close the loop.

5.3.3 Sensing

As we wish to learn control policies that map from observations to controls, the choice of observations are critical to successful learning. Each Phantom is equipped with 3 optical encoders at a resolution of 5K steps per radian. We use a low-pass filter to compute the joint velocities. We also rely on a Vicon motion capture system, which gives us position data at 240Hz for the object we are manipulating–we assume the object remains upright and do not include orientation. While being quite precise ($0.1mm$ error), the overall accuracy is

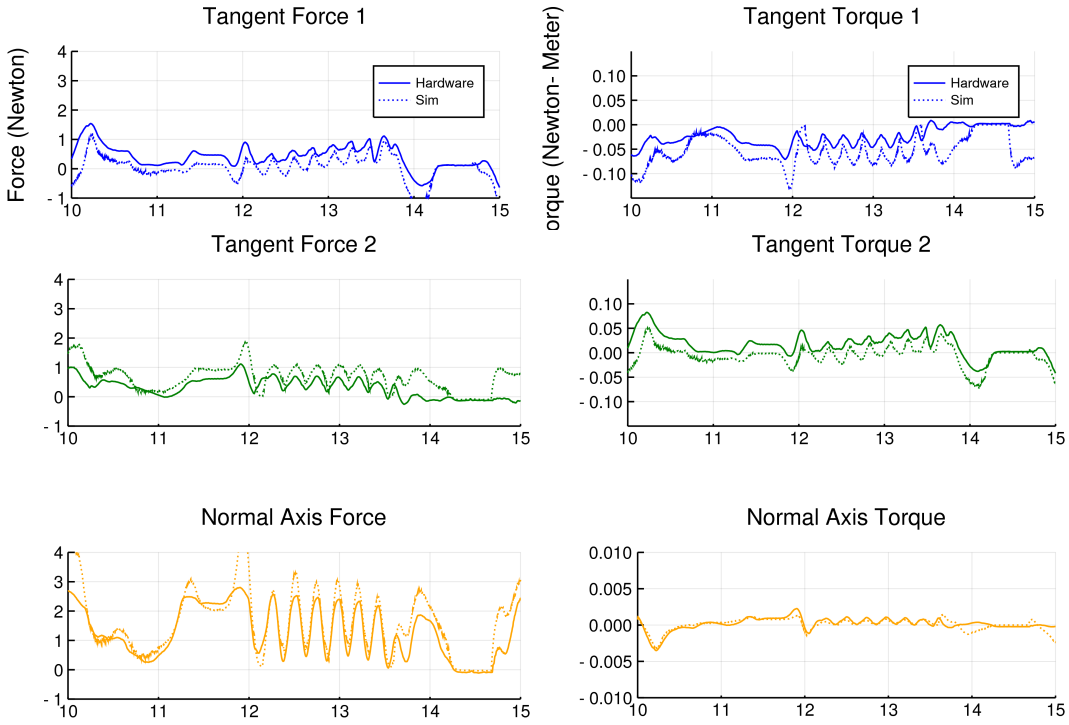


Figure 5.4: In these plots, we seed our simulator with the state measured from the hardware system. We use the simulator to calculate the instantaneous forces measured by a simulated force sensor, per time-step of real world data, and compare it to the data collected from hardware. These plots are in the frame of the contact, with the force along the normal axis being greatest. Note that the Y-axis of the Normal Axis Torque plot is different from the other torque plots.

significantly worse ($< 1mm$) due, in part, to imperfect object models and camera calibrations. While the Phantoms' joint position sensors are noiseless, they often have small biases due to imperfect calibration. One Phantom robot is equipped with an ATI Nano17 3-axis force/torque sensor. This data is not used during training or in any learned controller, but used as a means of hardware / simulation comparison described in a later section. The entire system is simulated for policy training in the MuJoCo physics engine [150].

In total, our control policy has an observational input of 36 dimensions with 9 actuator outputs– the 9 positions and 9 velocities of the Phantoms are converted to 15 positions and 15 velocities for modeling purposes due to the parallel linkages. We additionally use 3 positions for both the manipulated object and the tracked goal, with 9 outputs for the 3 actuators per Phantom. Velocity observations are not used for the object as this would require state estimation that we have deliberately avoided.

5.3.4 System Identification

System identification of model parameters was performed in our prior work [68], but modeling errors are difficult to eliminate completely. For system ID we collected various behaviors with the robots, ranging from effector motion in free space to infer intrinsic robot parameters, to manipulation examples such as touching, pushing and sliding between the end effector and the object to infer contact parameters. The resulting data is fed into the joint system ID and state estimation optimization procedure. As explained in [68], state estimation is needed when doing system ID in order to eliminate the small amounts of noise and biases that our sensors produce.

The recorded behaviors are represented as a list of sensor reading observations $\mathbf{O} = \{o_1, o_2, \ldots, o_n\}$ and motor torque action $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$. State estimation means finding a trajectory of states $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$. Each state is a vector $s_i = (\theta_1, \ldots, \theta_{k'}, x, y, z, q_w, q_x, q_y, q_z)$, representing joint angles and object position. We also perform system identification to find the set of parameters $\mathbf{P}$, which include coefficients of friction, contact softness, damping coefficients, link inertias and others. We then pose the system identification and estimation problems as a joint optimization problem:

$$\min_{\mathbf{P},\mathbf{S}} \sum_{i=1\ldots n} \|\hat{\tau}_i - a_i\|^{*_1} + \sum_{i=1\ldots n} \|\hat{o}_i - o_i\|^{*_2}$$

where $\hat{\tau}_i$ (predicted control signal) and $\hat{o}_i$ (predicted sensor outputs) are computed by the inverse dynamics generative model of MuJoCo: $(\hat{\tau}_i, \hat{s}_i) = \text{mj\_inverse}(q_{i-1}, q_i, q_{i+1})$. The optimization problem is solved via Gauss-Newton [68].

5.4 Task & Experiments

In this section we first describe the manipulation task used to evaluate learned policy performance, then describe the practical considerations involved in using the NPG algorithm in this work. Finally, we describe two experiments evaluating learned policy performance in both simulation and on hardware.

5.4.1 Task Description

We use the NPG algorithm to learn a pushing task. The goal is to reduce the distance of the object, in this case the cylinder, to a target position as much as possible. This manipulation task requires that contacts can be made and broken multiple times during a pushing movement. As there are no state constraints involved in this RL algorithm, we cannot guarantee that the object will reach the target location (the object can be pushed into an un-reachable location). We feel that this is an acceptable trade-off if we can achieve more robust control over a wider state space.

For these tasks we model the bases of each Phantom as fixed, arrayed roughly equilateral around the object being manipulated–this is to achieve closure around the object. We do not enforce a precise location for the bases to make the manipulation tasks more challenging and expect them to shift during operation regardless.

5.4.2 Training Considerations

Policy training is the process of discovering which actions the controller should take from the current state to achieve high reward. As such, it has implications for how well-performing the final policy is. Training structure informs the policy of good behavior, but is contrasted with the time required to craft the reward function. In this task's case, we use a very simple reward structure. In addition to the primary reward of reducing the distance between the object and the goal location, we provide the reward function with terms to reduce the distance between each finger tip and the object. This kind of hint term is common in both reinforcement

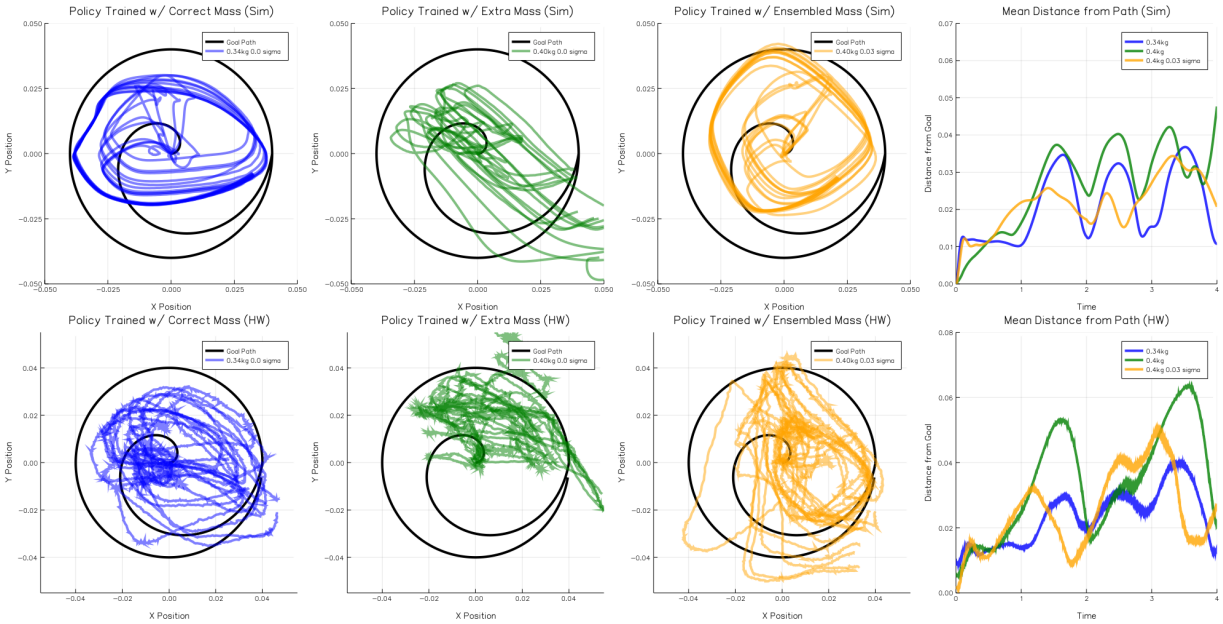


Figure 5.5: 10 rollouts are performed where the target position of the object is the path that spirals from the center outward (in black) and then performs a full circular sweep (the plots represent a top-down view). We compare three differently trained policies: one where the mass of the object cylinder is 0.34kg, one where the mass is increased by 20 percent (to 0.4kg), and finally, we train a policy with the incorrect mass, but add model noise (at standard deviation 0.03) during training to create an ensemble. We evaluate these policies on the correct (0.34kg) mass in both simulation and on hardware. In both, the policy trained with the incorrect mass does not track the goal path, sometimes even losing the object. We also calculate the per time-step error from the goal path, averaged from all 10 rollouts (right-most plots); there is usually a non-zero error between the object and the reference due to the feedback policy having to 'catch up' to the reference.

learning and trajectory optimization. There is also a control cost, a, that penalizes using too

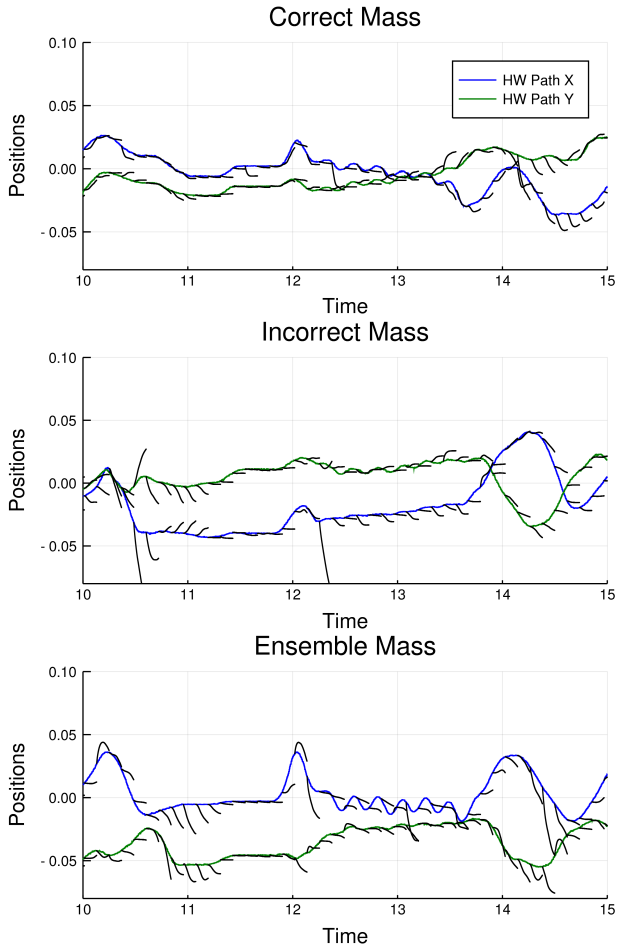


Figure 5.6: We show the effects of different controllers. We seed our simulator with the hardware's state, and, in simulation, perform a rollout of 200 time-steps – about 0.1 seconds. This is rendered as the black lines above. The correct policy (trained with measured mass), has rollouts that closely match the measured hardware state data. The incorrect policy (trained with an incorrect mass), performs differently in simulation. The remaining ensemble policy performs better than the incorrect one; this demonstrates that a 'safe' policy can be learned to at least begin an iterative data collection process. While it could be expected that the policies perform similarly in both simulation and hardware, we see that it is not the case here. A policy trained on an incorrect model would over-fit to the incorrect model.

much torque. The entire reward function at time t is as follows:

$$R_t(s,a) = 1 - 3\|O_{xy} - G_{xy}\| - \sum_{i=1}^{3} \|f_i - O_{xy}\| - 0.1a^2 \tag{5.5}$$

Where O_{xy} is the current position on the xy-plane of the object, G_{xy} is the goal position, and f_i is the Phantom end effector position. The state s consists of joint positions, velocities, and goal position. The actions (a) are torques.

The initial state of each trajectory rollout is with the Phantom robots at randomized joint positions, deliberately not contacting the cylinder object. The cylinder location is kept at the origin on the xy-plane, but the desired goal location is set to a uniformly random point within a 12cm diameter circle around the origin. To have more diverse initializations and to encourage robust policies, new initial states have a chance of starting at some state from one of the previous iteration's trajectories, provided the previous trajectory had a high reward. If the initial state was a continuation of the previous trajectory, the target location was again randomized: performing well previously only gives an initial state, and the policy needs to learn to push the object to a new goal. This is similar to a procedure outlined in [86] for training interactive and robust policies.

Finally, to further encourage robust behavior, we vary the location of the base of the Phantom robots by adding Gaussian noise before each rollout. The standard deviation of this noise is 0.5cm. In this way an ensemble of models is used in discovering robust behavior. As discussed above, we expect to have imprecisely measured their base locations and for each base to potentially shift during operation. We examine this effect more closely as one of our experiments.

5.4.3 Experiments

We devised two experiments to explore the validity of the NPG reinforcement learning algorithm to discover robust policies for difficult robotic manipulation tasks.

First, we collect runtime data of positions, velocities, and force-torque measurements from

a sensor equipped Phantom using the best performing controller we have learned. We use this hardware data (positions and calculated velocities of the system) to seed our simulator, where instantaneous forces are calculated using a simulated force-torque sensor. This data is compared to force-torque data collected from the hardware. Instantaneous force differences highlight the inaccuracies between a model in simulation and data in the real world that eventually lead to divergence.

We also compare short trajectory rollouts in simulation that have been seeded with data collected with hardware. This compares the policy's behavior, not the system's, as we wish to examine the performance of the policy in both hardware and simulation. Ideally, if the simulated environment matches the hardware, it can be taken as an indication that the system identification has been performed correctly. Secondly, we would like to compare the behavior of the learned control policy. From the perspective of task completion, the similarity or divergence of sim and real is less important as long as the robot completes the tasks satisfactorily. Said another way, poor system identification or sim/real divergence matters less if the robot gets the job done. For these experiments, the target location is set by the user by moving a second tracked object above the cylinder. This data was recorded and used to collect the above datasets for analysis.

As a second experiment direction, we show how the effects of model ensembles during training affect robustness and feasibility. To do this, we explicitly vary the mass of the object being manipulated. The object (cylinder) was measured to be 0.34kg in mass, therefore, we train a policy with the mass set to this value. The object's mass was chosen to be modified due to the very visible effects an incorrect mass would have on performance. We train two additional policies, both with a mass of 0.4kg (approximately 20% more mass). One of the additional policies is trained with an approximated ensemble: we add Gaussian noise to the object mass parameter with standard deviation of 0.03 (30 grams). All three policies are evaluated in simulation with a correctly measured object mass, and in the real world with our 0.34kg cylinder.

To evaluate the policies, we calculate a path for the target to follow. The path is a spiral

from the origin moving outward until it achieves a radius from the origin of 4 cm, at which it changes to a circular path and makes a full rotation, still at a radius of 4cm. This takes 4 seconds to complete. This path was programmatically set in both the simulator and on the real hardware to be consistent. This object ideally follows this trajectory path, as it presents a very visible means to explore policy performance.

Table 5.1: Average Distance from Target, 10 Rollouts

	Sim	Hardware
0.34kg Policy (correct)	2.1cm	2.33cm
0.4kg Policy (incorrect)	2.65cm	3.4cm
0.4kg Policy ensemble	2.15cm	2.52cm

5.5 Results

The results for the two experiments are presented as follows, with additional discussion in the next section.

5.5.1 Simulation vs Hardware

We show comparisons between calculated forces and torques in simulation and hardware in figure 5.4. Our simulated values closely match the sensed hardware values. However, the discretization of hardware sensors for the joint positions and velocities are not as precise as simulation, which may result in a different calculation of instantaneous forces. While MuJoCo can represent soft contacts, the parameters defining them were not identified accurately. Critically, we can see that when contact is not being made, the sensors, simulated and hardware, are in agreement.

We find that our learned controllers are still able to perform well at task completion, despite differences between simulation and hardware. We can see in figure 5.6 that for the correct policy (learned with a correct model), when we perform a rollout in simulation based on hardware data, the simulated rollout is close to the data collected from hardware. The policy performance in simulation is close to the policy performance in hardware. This is not the case for the policy trained using incorrect mass and the ensemble policy, where the simulated rollout is different from the hardware data. We hypothesize that policies trained on specific models over-fit to these models, and take advantage of the specifics of the model to complete the task. Despite the correct simulation being similar to hardware, the controller's behavior could cause divergence on whatever remaining small parameter differences. The ensemble policy, as expected, lies somewhere between the correct and incorrect policy (trained with incorrect parameters).

5.5.2 Training with Ensembles

We find that training policies with model ensembles to be particularly helpful. Despite being given a very incorrect mass of the object, the policy trained with the ensemble performed very well (figure 5.5). In addition to performing well in simulation, we found it to perform nearly as well as the correctly trained policy on hardware. Given that these are learned feedback controllers, there is a distinct lag of the object behind the desired reference trajectory. Table 5.1 quantifies the error from the reference trajectory across the whole length of the action. This mirrors our comparison in the previous section, where the correct policy performs comparably in both hardware and simulation, with the other policies less so. This is important to note given the poor performance of the incorrect policy: this task's training is indeed sensitive to this model parameter. The implication of the ensemble approach is not just that it can overcome poor or incorrect modeling, but can provide a safe initial policy to collect valuable data to improve the model.

5.6 Conclusion

Our results suggest two interesting observations. Simulation can provide a safe backdrop to test and develop non-intuitive controllers (see figure 5.3). This controller was developed for a robotic system without an intermediate controller such as PID, and without human demonstrations to shape the behavior. We also eschewed the use of a state estimator during training and run-time as this would add additional modeling reliance and complexity. Our simulation based policy learning approach also conveniently allows for building robust controllers by creating ensembles of models by varying physical parameters.

We show how simulated ensemble methods provide two major benefits. Firstly, it can partially make up for incorrectly measured / identified model parameters. This benefit should be obvious: it can be difficult to measure model parameters affecting nonlinear physical phenomena. Additionally, training in an ensemble has the added benefit of allowing for more conservative policies to enable appropriate data collection for actual model improvement. A natural extension of this observation would be full model adaptation using a technique such as EPopt [118].

Model adaptation provides a bridge between model-based and model-free methods. Leveraging a model in simulation can provide a useful policy to begin robot operation, which can subsequently be fine-tuned on hardware in a model-free mode. Very dynamic behaviors may not be suited to direct hardware training without significant human imposed safety constraints, which may take significant time to develop and may not account for all use cases. Given that most robots are manufactured using modern techniques, a model to be used in simulation is very likely to exist, and this model should be leveraged to obtain better policies.

Chapter 6

ONLINE OPTIMIZATION WITH LEARNING

Autonomously learning intelligent behavior for complex control problems has been a challenge due to the large search space of these high dimensional problems. While possible to accelerate the process by employing model based control, these methods cannot leverage experience to improve over time. On the other hand, model free reinforcement learning techniques are fully susceptible to the curse of dimensionality and struggle with exploration. We propose a "plan online and learn offline" framework for the setting where an agent, with an internal model, needs to continually act and learn in the world. Our work builds on the synergistic relationship between local model-based control, global value function learning, and exploration. We study how local trajectory optimization can cope with approximation errors in the value function, and can stabilize and accelerate value function learning. Conversely, we also study how approximate value functions can help reduce the planning horizon and allow for better policies beyond local solutions. Finally, we also demonstrate how trajectory optimization can be used to perform temporally coordinated exploration in conjunction with estimating uncertainty in value function approximation. This exploration is critical for fast and stable learning of the value function. Combining these components enable solutions to complex control tasks, like humanoid locomotion and dexterous in-hand manipulation, in the equivalent of a few minutes of experience in the real world.

We consider a setting where an agent with limited memory and computational resources is dropped into a world. The agent has to simultaneously act in the world and learn to become proficient in the tasks it encounters. Let us further consider a setting where the agent has some prior knowledge about the world in the form of a nominal dynamics model. However, the state space of the world could be very large and complex, and the set of possible tasks

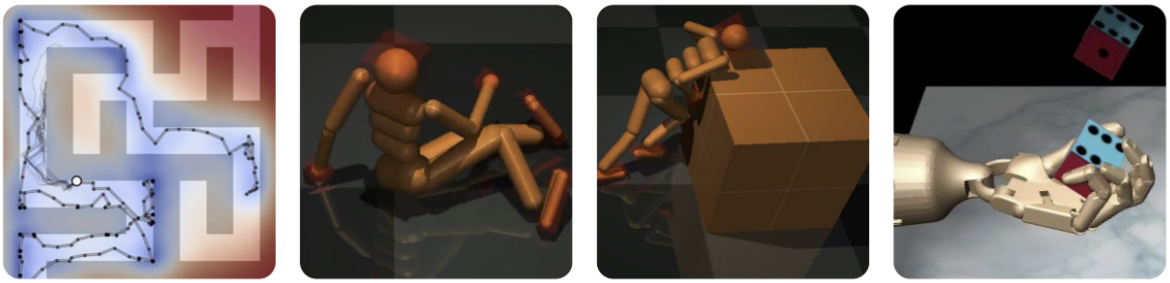


Figure 6.1: Examples of tasks solved with POLO. A 2D point agent navigating a maze without any directed reward signal, a complex 3D humanoid standing up from the floor, pushing a box, and in-hand re-positioning of a cube to various orientations with a five-fingered hand. Video demonstration of our results can be found at: `https://sites.google.com/view/polo-mpc`.

very diverse. This complexity and diversity, combined with limited computational capability, rules out the possibility of an *omniscient* agent that has experienced all situations and knows how to act optimally in all states, even if the agent knows the dynamics. Thus, the agent has to act in the world while learning to become competent.

Based on the knowledge of dynamics and its computational resources, the agent is imbued with a local search procedure in the form of trajectory optimization. While the agent would certainly benefit from the most powerful of trajectory optimization algorithms, it is plausible that very complex procedures are still insufficient or inadmissible due to the complexity or inherent unpredictability of the environment. Limited computational resources may also prevent these powerful methods from real-time operation. While the trajectory optimizer may be insufficient by itself, we show that it provides a powerful vehicle for the agent to explore and learn about the world.

Due to the limited capabilities of the agent, a natural expectation is for the agent to be moderately competent for new tasks that occur infrequently and skillful in situations that it encounters repeatedly by learning from experience. Based on this intuition, we propose the *plan online and learn offline (POLO)* framework for continual acting and learning. POLO is

based on the tight synergistic coupling between local trajectory optimization, global value function learning, and exploration.

We will first provide intuitions for why there may be substantial performance degradation when acting *greedily* using an approximate value function. We also show that value function learning can be accelerated and stabilized by utilizing trajectory optimization integrally in the learning process, and that a trajectory optimization procedure in conjunction with an approximate value function can compute near optimal actions. In addition, exploration is critical to propagate global information in value function learning, and for trajectory optimization to escape local solutions and saddle points. In POLO, the agent forms hypotheses on potential reward regions, and executes temporally coordinated action sequences through trajectory optimization. This is in contrast to strategies like $\epsilon-$greedy and Boltzmann exploration that explore at the granularity of individual timesteps. The use of trajectory optimization enables the agent to perform directed and efficient exploration, which in turn helps to find better global solutions.

The setting studied in the paper models many problems of interest in robotics and artificial intelligence. Local trajectory optimization becomes readily feasible when a nominal model and computational resources are available to an agent, and can accelerate learning of novel task instances. In this work, we study the case where the internal nominal dynamics mo del used by the agent is accurate. Nominal dynamics models based on knowledge of physics [150], or through learning [80], complements a growing body of work on successful simulation to reality transfer and system identification [122, 118, 82, 98]. Combining the benefits of local trajectory optimization for fast improvement with generalization enabled by learning is critical for robotic agents that live in our physical world to continually learn and acquire a large repertoire of skills.

6.1 The POLO framework

The POLO framework combines three components: local trajectory optimization, global value function approximation, and an uncertainty and reward aware exploration strategy. We first

present the motivation for each component, followed by the full POLO procedure.

6.1.1 Definitions, Notations, and Setting

We model the world as an infinite horizon discounted Markov Decision Process (MDP), which is characterized by the tuple: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma\}$. $\mathcal{S} \in \mathbb{R}^n$ and $\mathcal{A} \in \mathbb{R}^m$ represent the continuous (real-valued) state and action spaces respectively. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represents the reward function. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ represents the dynamics model, which in general could be stochastic, and $\gamma \in [0, 1)$ is the discount factor. A policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ describes a mapping from states to actions. The value of a policy at a state is the average discounted reward accumulated by following the policy from the state: $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s]$. The overall performance of the policy over some start state distribution β is given by: $J^\beta(\pi) = \mathbb{E}_{s \sim \beta}[V^\pi(s)]$. For notational simplicity, we use s' to denote the next state visited after (from) s.

As described earlier, we consider the setting where an agent is dropped into a complex world. The agent has access to an internal model of the world. However, the world can be complex and diverse, ruling out the possibility of an omniscient agent. To improve its behavior, the agent has to explore and understand relevant parts of the state space while it continues to act in the world. Due to the availability of the internal model, the agent can revisit states it experienced in the world and reason about alternate potential actions and their consequences to learn more efficiently.

6.1.2 Value Function Approximation

The optimal value function describes the long term discounted reward the agent receives under the optimal policy. Defining the Bellman operator at state s as:

$$\mathcal{B}V(s) = \max_a \mathbb{E}\left[r(s, a) + \gamma V(s')\right], \tag{6.1}$$

the optimal value function V^* corresponds to the fixed point: $V^*(s) = \mathcal{B}V^*(s)\ \forall s \in \mathcal{S}$. For small, tabular MDPs, classical dynamic programming algorithms like value iteration

can be used to obtain the optimal value function. The optimal policy can be recovered from the value function as: $\pi^*(s) = \arg\max_a \mathbb{E}[r(s,a) + \gamma V^*(s')]$. For more complex MDPs, computing the optimal value function exactly is not tractable except in a few well known cases like the LQR [5] and LMDPs [149, 39]. Thus, various approximate techniques have been considered in prior works. One popular approach is fitted value iteration [18, 93], where a function approximator (e.g. neural network) is used to approximate the optimal value function. The core structure of fitted value iteration considers a collection of states (or a sampling distribution ν), and a parametric value function approximator $\hat{V}_\theta$. Inspired by value iteration, fitted value iteration updates parameters as:

$$\theta_{i+1} = \arg\min_\theta \ \mathbb{E}_{s\sim\nu}\left[\left(\hat{V}_\theta(s) - \mathcal{B}\hat{V}_{\theta_i}(s)\right)^2\right] \tag{6.2}$$

where $\mathcal{B}\hat{V}_{\theta_i}(s)$ are targets for the regression problem computed at the specific state s according to Eq. (6.1). After sufficient iterations of the procedure in Eq. (6.2) to get a good approximation, the policy is recovered as $\hat{\pi}(s) = \arg\max_a \mathbb{E}[r(s,a) + \gamma\hat{V}_\theta(s')]$. The success and convergence of this overall procedure depends critically on at least two components: the capacity and structure of the function approximator (θ); and the sampling distribution (ν).

Lemma 1. *[18] Let $\hat{V}$ be an approximate value function with ℓ_∞ error $\epsilon := \max_s |\hat{V}(s) - V^*(s)|$. Let $\hat{\pi}(s) = \arg\max_a \mathbb{E}[r(s,a) + \gamma\hat{V}(s')]$ be the induced greedy policy. For all MDPs and β, the bound in Eq. (6.3) holds. Furthermore, for any size of the state space, there exist MDPs and $\hat{V}$ for which the bound is tight (holds with equality).*

$$J^\beta(\pi^*) - J^\beta(\hat{\pi}) \leq \frac{2\gamma\epsilon}{1-\gamma} \tag{6.3}$$

Intuitively, this suggests that performance of $\hat{\pi}$ degrades with a dependence on effective problem horizon determined by γ. This can be understood as the policy paying a price of ϵ at every timestep. Due to the use of function approximation, errors may be inevitable. In practice, we are often interested in temporally extended tasks where $\gamma \approx 1$, and hence this possibility is concerning. Furthermore, the arg max operation in $\hat{\pi}$ could inadvertently exploit approximation errors to produce a poor policy. The performance of fitted value

iteration based methods also rely critically on the sampling distribution to propagate global information [93], especially in sparse reward settings. For some applications, it may be possible to specify good sampling distributions using apriori knowledge of where the optimal policy should visit (e.g. based on demonstration data). However, automatically generating such sampling distributions when faced with a new task may be difficult, and is analogous to the problem of exploration.

6.1.3 Trajectory Optimization and Model Predictive Control

Trajectory optimization and model predictive control (MPC) have a long history in robotics and control systems [47, 142][1]. In MPC, starting from state s_t and using the knowledge of the dynamics model, a locally optimal sequence of actions (or policies) up to a moving horizon of H is computed by solving the following optimization problem.

$$
\begin{aligned}
\underset{\{\tilde{\pi}_k\}_{k=t}^{t+H}}{\text{maximize}} \quad & \mathbb{E}\left[\sum_{k=t}^{t+H-1} \gamma^{(k-t)} r(\mathbf{x}_t, \mathbf{u}_t) + \gamma^H r_f(\mathbf{x}_{t+H})\right] && \text{(Trajectory Cost)} \\
\text{subject to} \quad & \mathbf{x}_{k+1} \sim \mathcal{T}(\mathbf{x}_k, \mathbf{u}_k) && \text{(Dynamics)} \\
& \mathbf{u}_k \sim \tilde{\pi}_t(\cdot|\mathbf{x}_k) && \text{(Local Policy)} \\
& \mathbf{x}_t = s_t && \text{(Initial State)} \\
\text{Result} \quad & a_t \sim \hat{\pi}_{MPC}(\cdot|s_t) \equiv \tilde{\pi}_t^*(\cdot|\mathbf{x}_t) && \text{(Solution to Optimization)}
\end{aligned}
\tag{6.4}
$$

Here, we use $\mathbf{x}, \mathbf{u}, \tilde{\pi}$ as dummy variables for states, actions, and policy to distinguish the "imagined" evolution of the MDP used for the trajectory optimization with the actual states (s) observed in the true evolution of the MDP. Here, $r(\mathbf{x}, \mathbf{u})$ represents the running reward which is the same as the MDP reward function, and $r_f(\mathbf{x}_{t+H})$ represents a terminal reward function. Let $\{\tilde{\pi}_k^*\}$ be the local time-indexed policies obtained as the solution to the optimization problem in (6.4). After solving the optimization problem, the first local time-indexed policy is used as $\hat{\pi}_{MPC}(\cdot|s_t) \coloneqq \tilde{\pi}_t^*(\cdot|\mathbf{x}_t)$. The entire procedure is repeated again

[1] In this chapter, we use the terms trajectory optimization and MPC interchangeably

in the next time step $(t+1)$. Note that we have defined the optimization problem over a sequence of feedback policies. However, if the dynamics is deterministic, a sequence of actions $\{\mathbf{u}_k\}_{k=t}^{t+H}$ can be optimized and used instead without any loss in performance. See Appendix 6.3 for further discussions. This approach has led to tremendous success in a variety of control systems such as power grids, chemical process control [114], and more recently in robotics [155]. Since MPC looks forward only H steps, it is ultimately a local method unless coupled with a value function that propagates global information. In addition, we also provide intuitions for why MPC may help accelerate the learning of value functions. This synergistic effect between MPC and global value function forms a primary motivation for POLO.

Impact of approximation errors in the value function

Lemma 2. *Let $\hat{V}$ be an approximate value function with ℓ_∞ error $\epsilon := \max_s |\hat{V}(s) - V^*(s)|$. Suppose the terminal reward in Eq. (6.4) is chosen as $r_f(s_H) = \hat{V}(s_H)$, and let the MPC policy be $\hat{\pi}_{MPC}(\cdot|s_t) \coloneqq \tilde{\pi}_t^*(\cdot|\mathbf{x}_t)$ (from Eq. 6.4). Then, for all MDPs and β, the performance of the MPC policy can be bounded as:*

$$J^\beta(\pi^*) - J^\beta(\hat{\pi}_{MPC}) \leq \frac{2\gamma^H \epsilon}{1-\gamma^H}. \tag{6.5}$$

Proof. The proof is provided in appendix 6.3 at the end of the chapter. □

This suggests that MPC (with $H > 1$) is less susceptible to approximation errors than greedy action selection. Also, without a terminal value function, we have $\epsilon = \mathcal{O}(r_{\max}/(1-\gamma))$ in the worst case, which adds an undesirable scaling with the problem horizon.

Accelerating convergence of the value function Furthermore, MPC can also enable faster convergence of the value function approximation. To motivate this, consider the H-step Bellman operator: $\mathcal{B}^H V(s) := \max_{a_{0:H-1}} \mathbb{E}[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V(s_H)]$. In the tabular setting, for any V_1 and V_2, it is easy to verify that $|\mathcal{B}^H V_1 - \mathcal{B}^H V_2|_\infty \leq \gamma^H |V_1 - V_2|_\infty$. Intuitively, $\mathcal{B}^H$ allows for propagation of global information for H steps, thereby accelerating the convergence

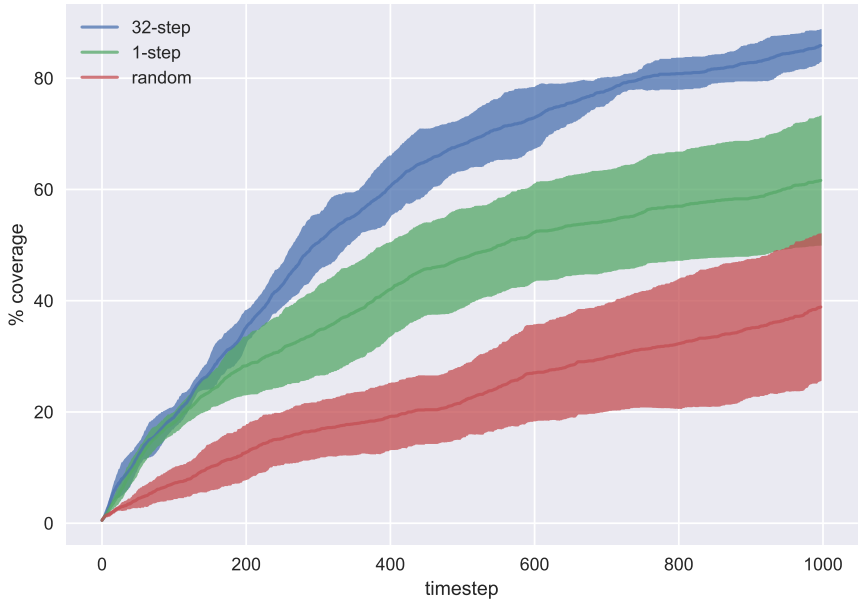


Figure 6.2: Area coverage of a 2D point mass navigation task in a world with *no rewards*. This figure describes the percentage of an occupancy grid covered by the agent, averaged over 10 random seeds

due to faster mixing. Note that one way to realize $\mathcal{B}^H$ is to simply apply $\mathcal{B}$ H times, with each step providing a contraction by γ. In the general setting, it is unknown if there exists alternate, cheaper ways to realize $\mathcal{B}^H$. However, for problems in continuous control, MPC based on local dynamic programming methods [57, 151] provide an efficient way to *approximately* realize $\mathcal{B}^H$, which can be used to accelerate and stabilize value function learning.

6.1.4 Planning to Explore

The ability of an agent to explore the relevant parts of the state space is critical for the convergence of many RL algorithms. Typical exploration strategies like ϵ-greedy and Boltzmann take exploratory actions with some probability on a *per time-step basis.* Instead, by using MPC, the agent can explore in the space of trajectories. The agent can consider a hypothesis of potential reward regions in the state space, and then execute the optimal

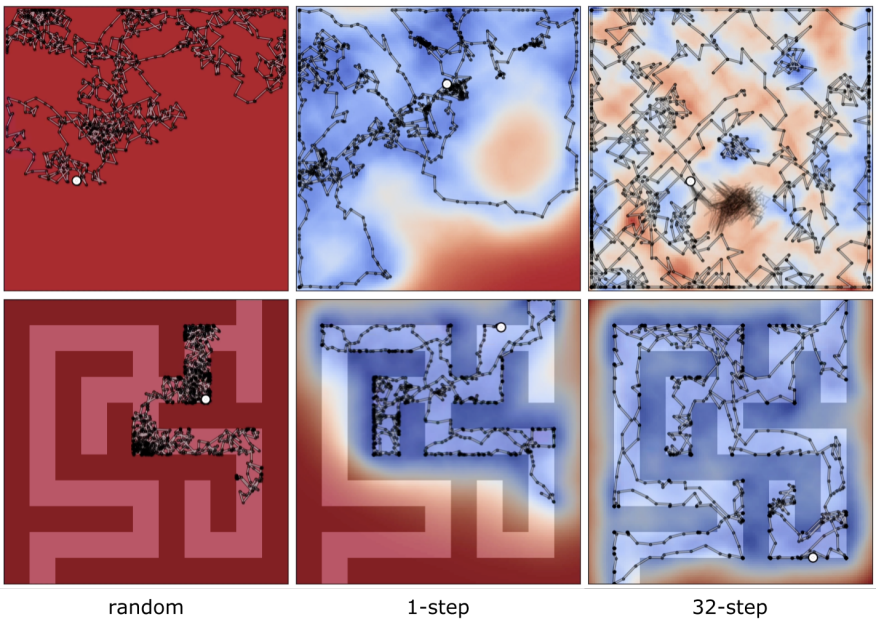


Figure 6.3: Exploration trace of a 2D point mass navigation task in a world with *no rewards*. The image depicts an agent over 1000 timesteps; red indicates regions of high value (uncertainty) while blue denotes low. The value function learns to assign the true, low values to regions visited and preserves high values to unexplored regions; uncertainty and long horizons are observed to be critical for exploration.

trajectory conditioned on this belief, resulting in a temporally coordinated sequence of actions. By executing such coordinated actions, the agent can cover the state space more rapidly and intentionally, and avoid back and forth wandering that can slow down the learning. We demonstrate this effect empirically in Section 6.2.1.

To generate the hypothesis of potentially rewarding regions, we take a Bayesian view and approximately track a posterior over value functions. Consider a motivating setting of regression, where we have a parametric function approximator f_θ with prior $\mathbb{P}(\theta)$. The dataset consists of input-output pairs: $\mathcal{D} = (x_i, y_i)_{i=1}^n$, and we wish to approximate $\mathbb{P}(\theta|\mathcal{D})$. In the Bayesian linear regression setting with Gaussian prior and noise models, the solution

to the following problem generates samples from the posterior [100, 8, 99]:

$$\arg\min_{\theta} ||\tilde{y}_i - f_{\tilde{\theta}}(x_i) - f_\theta(x_i)||_2^2 + \frac{\sigma^2}{\lambda}||\theta||_2^2 \tag{6.6}$$

where $\tilde{y}_i \sim \mathcal{N}(y_i, \sigma^2)$ is a noisy version of the target and $\tilde{\theta} \sim \mathbb{P}(\theta)$ is a sample from the prior. Based on this, [99] demonstrate the benefits of uncertainty estimation for exploration. Similarly, we use this procedure to obtain samples from the posterior for value function approximation, and utilize them for temporally coordinated action selection using MPC. We consider K value function approximators $\hat{V}_\theta$ with parameters $\theta_1, \theta_2, \ldots \theta_K$ independently trained based on Eq. (6.6). We consider the softmax of the different samples as the value at a state:

$$\hat{V}(s) = \sum_{k=1}^{K} \omega_k(s) \hat{V}_{\theta_k}(s), \quad \text{where} \quad \omega_k(s) \stackrel{\text{def}}{:=} \frac{\exp\left(\kappa \hat{V}_{\theta_k}(s)\right)}{\sum_{j=1}^{K} \exp\left(\kappa \hat{V}_{\theta_j}(s)\right)} \tag{6.7}$$

Since the above scheme approximates mean + variance for small $\kappa > 0$, this procedure encourages the agent to additionally explore parts of the state space where the disagreement between the function approximators is large. This corresponds to the broad notion of optimism in the face of uncertainty [7] which has been successful in a number of applications [42, 78].

6.1.5 Final Algorithm

To summarize, POLO utilizes a global value function approximation scheme, a local trajectory optimization subroutine, and an optimistic exploration scheme. POLO operates as follows: when acting in the world, the agent uses the internal model and always picks the optimal action suggested by MPC. Exploration is implicitly handled by tracking the value function uncertainties and the optimistic evaluation, as specified in Eq. (6.6) and (6.7). All the experience (visited states) from the world are stored into a replay buffer $\mathcal{D}$, with old experiences discarded if the buffer becomes full. After every Z steps of acting in the world and collecting experience, the value functions are updated by: (a) constructing the targets according to Eq. (6.8); (b) performing regression using the randomized prior scheme using Eq. (6.6) where f_θ corresponds to the value function approximator. For state s in the buffer

Algorithm 3 Plan Online and Learn Offline (POLO)

Inputs: planning horizon H, value function parameters $\theta_1, \theta_2, \ldots \theta_K$, mini-batch size n, number of gradient steps G, update frequency Z
for $t = 1$ **to** ∞ **do**
  Select action a_t according to MPC (Eq. 6.4) with terminal reward $r_f(s) \equiv \hat{V}(s)$ from Eq. (6.7)
  Add the state experience s_t to replay buffer $\mathcal{D}$
  if $\text{mod}(t, Z) = 0$ **then**
    for G times **do**
      Sample n states from the replay buffer, and compute targets using Eq. (6.8)
      Update the value functions using Eq. (6.6) (see Section 6.1.5 for details)
    end for
  end if
end for

and value network k with parameters θ_k, the targets are constructed as:

$$y^k(s) = \max_{\{\tilde{\pi}_t\}_{t=0}^{N-1}} \mathbb{E}\left[\sum_{t=0}^{N-1} \gamma^t r(\mathbf{x}_t, \mathbf{u}_t) + \gamma^N \hat{V}_{\theta_k}(\mathbf{x}_N)\right], \text{ where } \mathbf{x}_0 = s, \mathbf{u}_t \sim \tilde{\pi}_t(\cdot|\mathbf{x}_t) \tag{6.8}$$

which corresponds to solving a $N-$step trajectory optimization problem starting from state s. As described earlier, using trajectory optimization to generate the targets for fitting the value approximation accelerates the convergence and makes the learning more stable, as verified experimentally in Section 6.2.3. The overall procedure is summarized in Algorithm 3.

6.2 Empirical Results and Discussion

Through empirical evaluation, we wish to answer the following questions:

1. Does trajectory optimization in conjunction with uncertainty estimation in value function approximation result in temporally coordinated exploration strategies?

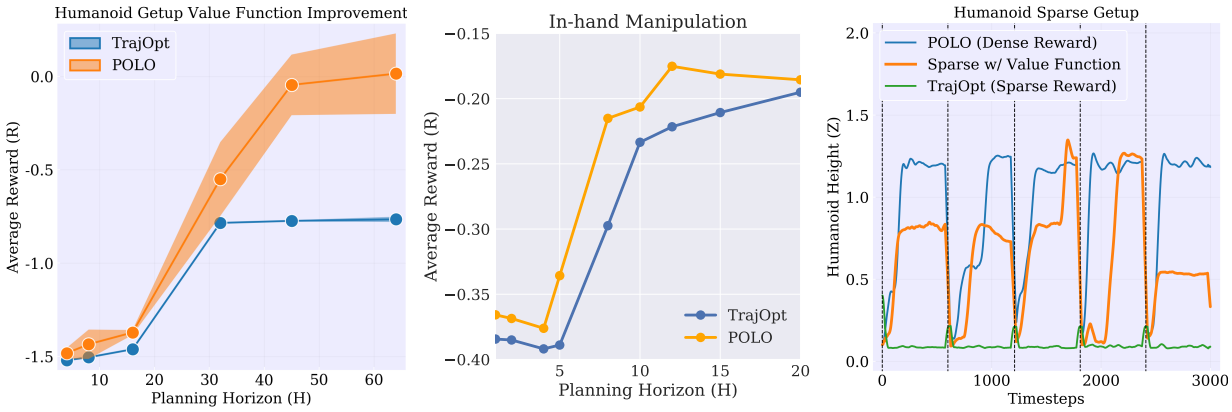


Figure 6.4: Performance as a function of planning horizon for the humanoid getup (left), and in-hand manipulation task (middle). POLO was trained for 12000 and 2500 environment timesteps, respectively. We test POLO with the learned terminal value function against pure MPC and compare average reward obtained over 3 trials in the getup task and 1000 steps in the manipulation task. On the right, a value function trained with POLO is used by MPC without per-time-step rewards. The agent's height increases, indicating a task-relevant value function. For comparison, we also include the trace of POLO with dense rewards and multiple trials (dashed vertical lines)

2. Can the use of an approximate value function help reduce the planning horizon for MPC?

3. Does trajectory optimization enable faster and more stable value function learning?

Before answering the questions in detail, we first point out that POLO can scale up to complex high-dimensional agents like 3D humanoid and dexterous anthropomorphic hand [98, 120] which are among the most complex control tasks studied in robot learning. Video demonstration can be found at: `https://sites.google.com/view/polo-mpc`

6.2.1 Trajectory optimization for exploration

Exploration is critical in tasks where immediate rewards are not well aligned with long-term objectives. As a representative problem, we consider a point mass agent in different 2D worlds illustrated in figure 6.3: a simple finite size box with no obstacles and a maze. This domain serves to provide an intuitive understanding of the interaction between trajectory optimization and exploration while also enabling visualization of results. In the extreme case of no rewards in the world, an agent with only local information would need to continuously explore. We wish to understand how POLO, with its ensemble of value functions tracking uncertainties, uses MPC to perform temporally coordinated actions. Our baseline is an agent that employs random exploration on a per-time-step basis; MPC without a value function would not move due to lack of local extrinsic rewards. Second, we consider an agent that performs uncertainty estimation similar to POLO but selects actions greedily (i.e. POLO with a planning horizon of 1). Finally, we consider the POLO agent which tracks value uncertainties and selects actions using a 32-step MPC procedure. We observe that POLO achieves more region coverage in both point mass worlds compared to alternatives, as quantitatively illustrated in figure 6.3(a). The ensemble value function in POLO allows the agent to recognize the true, low value of visited states, while preserving an optimistic value elsewhere. Temporally coordinated action is necessary in the maze world; POLO is able to navigate down all corridors.

6.2.2 Value function approximation for trajectory optimization

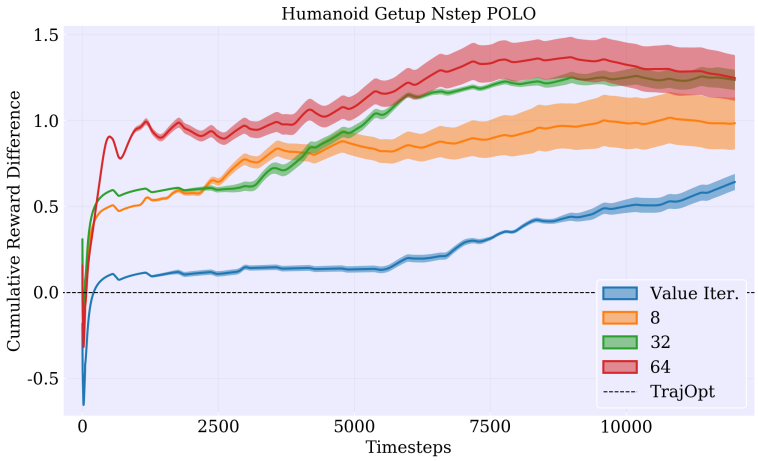
Next, we study if value learning helps to reduce the planning horizon for MPC. To this end, we consider two high dimensional tasks: *humanoid getup* where a 3D humanoid needs to learn to stand up from the ground, and *in-hand manipulation* where a five-fingered hand needs to re-orient a cube to a desired configuration that is randomized every 75 timesteps. For simplicity, we use the MPPI algorithm [155] for trajectory optimization. In Figure 6.4, we consider MPC and the full POLO algorithm of the same horizon, and compare their performance after T steps of learning in the world. We find that POLO uniformly dominates

MPC, indicating that the agent is consolidating experience from the world into the value function. With even the longest planning horizon, the humanoid getup task has a local solution where it can quickly sit up, but cannot discover a chain of actions required to stand upright. POLO's exploration allows the agent to escape the local solution, and consolidate the experiences to consistently stand up. To further test if the learned value function is task aligned, we take the value function trained with POLO, and use it with MPC *without any intermediate rewards*. Thus, the MPC is optimizing a trajectory of length $H = 64$ purely using the value function of the state after 64 steps. We observe, in Figure 6.4, that even in this case, the humanoid is able to consistently increase its height from the floor indicating that the value function has captured task relevant details. We note that a greedy optimization procedure with this value function does not yield good results, indicating that the learned value function is only approximate and not good everywhere.

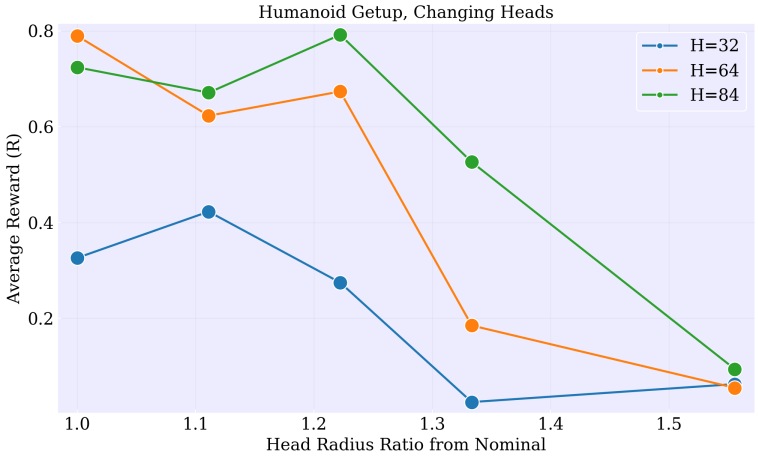
While the humanoid getup task presents temporal complexity requiring a large planning horizon, the in-hand manipulation task presents spatial complexity. A large number of time steps are not needed to manipulate the object, and a strong signal about progress is readily received. However, since the targets can change rapidly, the variance in gradient estimates can be very high for function approximation methods [50]. Trajectory optimization is particularly well suited for such types of problems, since it can efficiently compute near-optimal actions conditioned on the instance, facilitating function approximation. Note that the trajectory optimizer is unaware that the targets can change, and attempts to optimize a trajectory for a fixed instance of the task. The value function consolidates experience over multiple target changes, and learns to give high values to states that are not just immediately good but provide a large space of affordances for the possible upcoming tasks.

6.2.3 Trajectory optimization for value function learning

Finally, we study if trajectory optimization can aid in accelerating and stabilizing value function learning. To do so, we again consider the humanoid getup task and study different variants of POLO. In particular, we vary the horizon (N) used for computing the value



(a) POLO learning for different N-step horizons



(b) MPC with imperfect value function

Figure 6.5: Usefulness of trajectory optimization for value function learning. (a) illustrates that N-step trajectory optimization accelerates the learning of the value function. N=1 corresponds to trajectory centric fitted value iteration. A difference of 0.2 reward to MPC amounts to approximately 50% performance improvement. (b) value function trained for the nominal model (head size of 1.0) used with MPC for models with larger sizes.

function targets in Eq. (6.8). We observe that as we increase N, the agent learns the value function with fewer interactions with the world, as indicated in Figure 6.5(a). The benefit of using $N-$step returns for stable value function learning and actor-critic methods have been observed in numerous works [84, 92, 126], and our experiments reinforce these observations. The use of $N-$step returns help to traverse the bias-variance trade-off. Furthermore, due to the discounting, the contribution of $V(s_N)$ is made weaker and thus the targets are more stable. This mirrors ideas such as target networks [43] commonly used to stabilize training. As discussed earlier, longer horizons make trajectory optimization more tolerant to errors in the value function. To illustrate this, we take the value function trained with POLO on a nominal humanoid model, and perturb the model by changing the size of the head to model value function degradation. Figure 6.5(b) shows that a longer planning horizon can mitigate this degradation. This presents intriguing future possibility of using MPC to improve transfer learning between tasks or robot platforms.

6.3 Conclusions and Future Work

In this work we presented POLO, which combines the strengths of trajectory optimization and value function learning. In addition, we studied the benefits of planning for exploration in settings where we track uncertainties in the value function. Together, these components enabled control of complex agents like 3D humanoid and five-fingered hand. In this work, we assumed access to an accurate internal dynamics model. A natural next step is to study the influence of approximation errors in the internal model and improving it over time using the real world interaction data.

Appendix: Experimental Details, Humanoid

The model used for the humanoid experiments was originally distributed with the MuJoCo [150] software package and modified for our use. The model nominally has 27 degrees of freedom, including the floating base. It utilizes direct torque actuation for control, necessitating a small timestep of 0.008 seconds. The actuation input is limited to ± 1.0, but the

original gear ratios are left unchanged.

For POLO, the choice of inputs for the value function involves a few design decisions. We take inspiration from robotics by using only easily observed values.

Table 6.1: Dimensions and Parameters in Humanoid Experiments

Dims.	Observation
6	Direction & Normal Vector, Torso
3	Direction Vector, Neck to R. Hand
3	Direction Vector, Neck to L. Hand
3	Direction Vector, Hip to R. Foot
3	Direction Vector, Hip to L. Foot
5	Height, Root, Hands, & Feet
6	Root Velocities
5	Touch Sensors, Head, Hands, & Feet

Value	Parameter
0.99	γ discount Factor
64	Planning Horizon Length
120	MPPI Rollouts
0.2	MPPI Noise σ
1.25	MPPI Temperature

For value function approximation in POLO for the humanoid tasks, we use an ensemble of 6 neural networks, each of which has 2 layers with 16 hidden parameters each; *tanh* is used for non-linearity. Training is performed with 64 gradient steps on minibatches of size 32, using ADAM with default parameters, every 16 timesteps the agent experiences.

In scenarios where the agent resets, we consider a horizon of 600 timesteps with 20 episodes, giving a total agent lifetime of 12000 timesteps or 96 seconds. When we consider no resets, we use the same total timesteps. A control cost is shared for each scenario, where we penalize an actuator's applied force scaled by the inverse of the mass matrix. Task specific rewards are as follows.

Humanoid Getup

In the getup scenario, the agent is initialized in a supine position, and is required to bring its root height to a target of 1.1 meters. The reward functions used are as follows. In the non-sparse case, the difficulty in this task is eschewing the immediate reward for sitting in favor of the delayed reward of standing; this sequence is non-trivial to discover.

$$R(s) = \begin{cases} 1.0 - (1.25 - Root_z), & \text{if } Root_z \leq 1.25 \\ 1.0, & \text{otherwise} \end{cases}, R_{sparse}(s) = \begin{cases} 0.0, & \text{if } Root_z \leq 1.25 \\ 1.0, & \text{otherwise} \end{cases}$$

Humanoid Walk

In the walking scenario, the agent is initialized in an upright configuration. We specify a reward function that either penalizes deviation from a target height of 1.1 meters, or penalizes the deviation from both a target speed of 1.0 meters/second and the distance from the world's x-axis to encourage the agent to walk in a straight line. We choose a target speed as opposed to rewarding maximum speed to encourage stable walking gaits.

$$R(s) = \begin{cases} -(1.25 - Root_z), & \text{if } Root_z \leq 1.25 \\ 1.0 - |1.0 - Vel_x| - |Root_x|, & \text{otherwise} \end{cases}$$

Humanoid Box

For the box environment, we place a 0.9 meter wide cube in front of the humanoid, which needs to be pushed to a specific point. The friction between the box and ground is very low, however, and most pushes cause the box to slide out of reach; POLO learns to better limit the initial push to control the box to the target.

$$R(s) = \begin{cases} -(1.25 - Root_z), & \text{if } Root_z \leq 1.25 \\ 2.0 - ||Box_{xy} - Root_{xy}||_2, & \text{else if } |Box_{xy} - Root_{xy}|_2 > 0.8 \\ 4.0 - 2 * ||Box_{xy} - Target_{xy}||_2, & \text{otherwise} \end{cases}$$

In this setup, the observation vector increases with the global position of the box, and the dimensionality of the system increase by 6. The box initially starts 1.5 meters in front of the humanoid, and needs to be navigated to a position 2.5 meters in front of the humanoid.

Appendix: Experimental Details, Hand Manipulation

We use the Adroit hand model [70] and build on top of the hand manipulation task suite of [120]. The hand is position controlled and the dice is modeled as a free object with 3 translational degrees of freedom and a ball joint for three rotational degrees of freedom. The base of the hand is not actuated, and the agent controls only the fingers and wrist. The dice is presented to the hand initially in some randomized configuration, and the agent has to reorient the dice to the desired configuration. The desired configuration is randomized every 75 timesteps and the trajectory optimizer does not see this randomization. Thus the randomization can be interpreted as unmodelled external disturbances to the system. We use a simple reward function for the task:

$$R(s) = -0.5\ \ell_1(x_o, x_g) - 0.05\ \ell_{quat}(q_o, q_g),$$

where x_o and x_g are the Cartesian positions of the object (dice) and goal respectively. The goal location for the dice is a fixed position in space and is based on the initial location of the palm of the hand. ℓ_1 is the L1 norm. q_o and q_g are the orientation configurations of object and goal, respectively, and expressed as quaternions with ℓ_{quat} being the quaternion difference.

We use 80 trajectories in MPPI with temperature of 10. We use an ensemble of 6 networks with 2 layers and 64 units each. The value function is updated every 25 steps of interaction with the world, and we take 16 gradient steps each with a batch size of 16. These numbers were arrived at after a coarse hyperparameter search, and we expect that better hyperparameter settings could exist.

Proof of Lemma 2 and Remarks

Let $\hat{\tau}$ and τ^* represent the trajectories of length H that would be generated by applying $\hat{\pi}_{MPC}$ and π^* respectively on the MDP. Starting from some state s, we have:

$$V^*(s) - V^{\hat{\pi}_{MPC}}(s) = \mathbb{E}_{\tau^*}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V^{\hat{\pi}_{MPC}}(s_H)\right] \tag{6.9}$$

Adding and subtracting, $\mathbb{E}_{\hat{\tau}}[\sum_t \gamma^t r_t + \gamma^H V^*(s_H)]$, we have:

$$\begin{aligned} V^*(s) - V^{\hat{\pi}_{MPC}}(s) = \gamma^H \mathbb{E}_{\hat{\tau}}\left[V^*(s_H) - V^{\hat{\pi}_{MPC}}(s_H)\right] \\ + \mathbb{E}_{\tau^*}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V^*(s_H)\right] - \mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V^*(s_H)\right]. \end{aligned} \tag{6.10}$$

Since $\max_s |\hat{V}(s) - V^*(s)| = \epsilon$, we have:

$$\mathbb{E}_{\tau^*}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V^*(s_H)\right] \leq \mathbb{E}_{\tau^*}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H \hat{V}(s_H)\right] + \gamma^H \epsilon \tag{6.11}$$

$$\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H V^*(s_H)\right] \geq \mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H \hat{V}(s_H)\right] - \gamma^H \epsilon \tag{6.12}$$

Furthermore, since $\hat{\tau}$ was generated by applying $\hat{\pi}_{MPC}$ which optimizes the actions using $\hat{V}$ as the terminal value/reward function, we have:

$$\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H \hat{V}(s_H)\right] \geq \mathbb{E}_{\tau^*}\left[\sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H \hat{V}(s_H)\right] \tag{6.13}$$

using these bounds, we have:

$$\begin{aligned} V^*(s) - V^{\hat{\pi}_{MPC}}(s) &\leq \gamma^H \mathbb{E}_{\hat{\tau}}\left[V^*(s_H) - V^{\hat{\pi}_{MPC}}(s_H)\right] + 2\gamma^H \epsilon \\ &\leq 2\gamma^H \epsilon \left(1 + \gamma^H + \gamma^2 H + \ldots\right) \\ &\leq \frac{2\gamma^H \epsilon}{1 - \gamma^H} \end{aligned} \tag{6.14}$$

by recursively applying the first bound to $V^*(s_H) - V^{\hat{\pi}_{MPC}}(s_H)$. This holds for all states, and hence for any distribution over states.

Notes and Remarks: For Eq. (6.13) to hold in general, and hence for the overall bound to hold, we require that the actions are optimized in closed loop. In other words, MPC has to optimize over the space of feedback policies as opposed to open loop actions. Many commonly used MPC algorithms like DDP and iLQG [57, 151] have this property through the certainty equivalence principle for the case of Gaussian noise. For deterministic dynamics, which is the case for most common simulators like MuJoCo, Eq. (6.13) holds without the closed loop requirement. We summarize the different cases and potential ways to perform MPC below:

- In the case of deterministic dynamics, the optimal open loop trajectory and optimal local feedback policies have the same performance up to finite horizon H. Thus, any trajectory optimization algorithm, such as iLQG and MPPI can be used.

- In the case of stochastic dynamics with additive Gaussian noise, local dynamic programming methods like iLQG and DDP [151, 57] provide efficient ways to optimize trajectories. These approaches also provide local feedback policies around the trajectories which are optimal due to the certainty equivalence principle.

- In the case of general stochastic systems, various stochastic optimal control algorithms like path integral control [145] can be used for the optimization. These situations are extremely rare in robotic control.

Finally, we also note that Sun et al. [137] propose and arrive at a similar bound in the context of imitation learning and reward shaping. They however assume that a policy can simultaneously optimize the approximate value function over H steps, which may not be possible for a parametric policy class. Since we consider MPC which is a non-parametric method (in the global sense), MPC can indeed simultaneously optimize for H steps using $\hat{V}$.

Chapter 7

CONCLUSION

This thesis presents a sequence of work that culminates in a method combining online optimization with learning to better allow for agent and robot behavior. If we consider intelligent behavior to be at the intersection of an agent's intent and the dynamics of the world, then said agent needs to perform well initially, while also learning from experience to improve. As shown, the method presented here, POLO, performs well on different agent morphologies and tasks without significant parameter changes, and does not require significant tuning and shaping of the reward function. Additionally, POLO also incorporates the ability to explore based on what the agent has learned in its value function: as tasks become increasingly complex while an agent's initial capability stays the same, the ability to do this directed exploration will become more critical.

This method presents a joining of different method families, online optimization and learning, at a new level of scale and complexity. We consider this work useful evidence that combining these methods is fruitful, and as a stepping stone to future methods. First, the expectation of a dynamics model for the trajectory optimizer can be relaxed and instead learned online along-side the value function, which would mitigate a strong assumption. Secondly, the usage of neural networks as the value function approximator provided the most flexibility in developing this method. The lack of structure in neural networks, however, lead naturally to the question of whether more principled function approximators can be used, such as graphical networks. The reason is two-fold: by explicitly modelling the uncertainty of the task value with respect to well defined variables rather than approximating this uncertainty over multiple entire networks, one can construct causal hypothesis between the variables and the reward that the agent could test for exploration purposes. This would provide the agent

with more ability to *reason* about the interaction between its intent and the world. The second reason is that a more structured representation could allow for the construction of variables and their relationships over multiple tasks, or even in unsupervised settings, such as if we were to task the robot to learn its own dynamics. Much like how humans are able to operate over their internal knowledge graphs at different scales and resolutions, we could consider an agent that is able to evaluate its current context and intent, and leverage the most critical information from its experience to perform: this requires additional structure in how an agent's experience is stored.

As our cyberphysical systems become more sophisticate, we hope that algorithms that drive these systems also grow in sophistication to support the goal of autonomous, intelligent behavior that complements and augments our capabilities.

BIBLIOGRAPHY

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.

[2] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

[3] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pages 5360–5370, 2017.

[4] Taylor Apgar, Patrick Clary, Kevin Green, Alan Fern, and Jonathan W Hurst. Fast online trajectory optimization for the bipedal robot cassie. In *Robotics: Science and Systems*, 2018.

[5] Karl Johan Åström and Richard M. Murray. Feedback systems an introduction for scientists and engineers. 2004.

[6] Christopher G. Atkeson. Using local trajectory optimizers to speed up global optimization in dynamic programming. In *NIPS*, 1993.

[7] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[8] Kamyar Azizzadenesheli, Emma Brunskill, and Anima Anandkumar. Efficient exploration through bayesian deep q-networks. *CoRR*, abs/1802.04412, 2018.

[9] Kamyar Azizzadenesheli, Brandon Yang, Weitang Liu, Emma Brunskill, Zachary Chase Lipton, and Anima Anandkumar. Sample-efficient deep rl with generative adversarial tree search. *CoRR*, abs/1806.05780, 2018.

[10] J. Andrew Bagnell, Sham M. Kakade, Andrew Y. Ng, and Jeff G. Schneider. Policy search by dynamic programming. In *NIPS*, 2003.

[11] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[12] Samuel Barrett, Matthew E Taylor, and Peter Stone. Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010.

[13] Andrew G Barto. Reinforcement learning. In *Neural systems for control*, pages 7–30. Elsevier, 1997.

[14] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72:81–138, 1995.

[15] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

[16] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[17] M. Benallegue, A. Mifsud, and F. Lamiraux. Fusion of force-torque sensors, inertial measurements units and proprioception for a humanoid kinematics-dynamics observation. In *2015 IEEE-RAS International Conference on Humanoid Robots*, November 2015.

[18] Dimitri Bertsekas and John Tsitsiklis. *Neuro-dynamic Programming.* 1996.

[19] Dimitri P Bertsekas. Approximate dynamic programming. 2008.

[20] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

[21] A. Bicchi and V. Kumar. Robotic grasping and contact: a review. In *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00*, volume 1, pages 348–353 vol.1, 2000.

[22] Lars Blackmore, Masahiro Ono, Askar Bektassov, and Brian C Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *IEEE transactions on Robotics*, 26(3):502–517, 2010.

[23] Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252. IEEE, 2018.

[24] Michael Bloesch, Marco Hutter, Mark A Hoepflinger, Stefan Leutenegger, Christian Gehring, C David Remy, and Roland Siegwart. State estimation for legged robots-consistent fusion of leg kinematics and imu. *Robotics*, 17:17–24, 2013.

[25] Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[26] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

[27] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[28] Arthur Bryson, Y.-C Ho, and George Siouris. Applied optimal control: Optimization, estimation, and control. *Systems, Man and Cybernetics, IEEE Transactions on*, 9:366 – 367, 07 1979.

[29] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

[30] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[31] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[32] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3381–3388. IEEE, 2017.

[33] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

[34] Mark Cutler, Thomas J Walsh, and Jonathan P How. Reinforcement learning with multi-fidelity simulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3888–3895. IEEE, 2014.

[35] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

[36] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[37] Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[38] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

[39] Krishnamurthy Dvijotham and Emanuel Todorov. A unifying framework for linearly solvable control. In *UAI*, 2011.

[40] Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *Humanoids*, pages 292–299. IEEE, 2013.

[41] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. In *RSS*, 2011.

[42] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 2016.

[43] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[44] Maurice F Fallon, Matthew Antone, Nicholas Roy, and Seth Teller. Drift-free humanoid state estimation fusing kinematic, inertial and lidar sensing. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 112–119. IEEE, 2014.

[45] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

[46] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[47] Carlos E. Garcia, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice - a survey. *Automatica*, 25:335–348, 1989.

[48] Christian Gehring, Stelian Coros, Marco Hutler, Carmine Dario Bellicoso, Huub Heijnen, Remo Diethelm, Michael Bloesch, Péter Fankhauser, Jemin Hwangbo, Mark Hoepflinger, et al. Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot. *IEEE Robotics & Automation Magazine*, 23(1):34–43, 2016.

[49] Alborz Geramifard, Thomas J Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, and Jonathan P How. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451, 2013.

[50] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. In *ICLR*, 2018.

[51] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[52] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. In *ICLR*, 2017.

[53] Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

[54] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NIPS*, 2015.

[55] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[56] Fumiya Iida, Gabriel Gómez, and Rolf Pfeifer. Exploiting body dynamics for controlling a running quadruped robot. In *ICAR'05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pages 229–235. IEEE, 2005.

[57] David Jacobson and David Mayne. *Differential Dynamic Programming*. American Elsevier Publishing Company, 1970.

[58] Matthew Johnson, Brandon Shrewsbury, Sylvain Bertrand, Tingfan Wu, Daniel Duran, Marshall Floyd, Peter Abeles, Douglas Stephen, Nathan Mertins, Alex Lesman, et al. Team ihmc's lessons learned from the darpa robotics challenge trials. *Journal of Field Robotics*, 32(2):192–208, 2015.

[59] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[60] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[61] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by a simple three-dimensional inverted pendulum model. *Advanced Robotics*, 17(2):131–147, 2003.

[62] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 1, pages 239–246. IEEE.

[63] Sham M Kakade. A natural policy gradient. In *NIPS*, 2001.

[64] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

[65] Hajime Kimura, Kazuteru Miyazaki, and Shigenobu Kobayashi. Reinforcement learning in pomdps with function approximation. In *ICML*, volume 97, pages 152–160, 1997.

[66] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[67] Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[68] Svetoslav Kolev and Emanuel Todorov. Physically consistent state estimation and system identification for contacts. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 1036–1043. IEEE, 2015.

[69] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[70] Vikash Kumar. *Manipulators and Manipulation in high dimensional spaces*. PhD thesis, University of Washington, Seattle, 2016.

[71] Vikash Kumar, Abhishek Gupta, Emanuel Todorov, and Sergey Levine. Learning dexterous manipulation policies from experience and imitation. *CoRR*, abs/1611.05095, 2016.

[72] Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In *ICRA*, 2016.

[73] Arthur D Kuo. A simple model of bipedal walking predicts the preferred speed–step length relationship. *Journal of biomechanical engineering*, 123(3):264–269, 2001.

[74] Nikolaos Kyriazis and Antonis Argyros. Physically plausible 3d scene tracking: The single actor hypothesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9–16, 2013.

[75] Gilwoo Lee, Siddhartha S Srinivasa, and Matthew T Mason. Gp-ilqg: Data-driven robust optimal control for uncertain nonlinear dynamical systems. *arXiv preprint arXiv:1705.05344*, 2017.

[76] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(39):1–40, 2016.

[77] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, 2013.

[78] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, 2010.

[79] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[80] Lennart Ljung. *System identification: theory for the user*. 1987.

[81] Kendall Lowrey, Jeremy Dao, and Emanuel Todorov. Real-time state estimation with whole-body multi-contact dynamics: a modified ukf approach. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 1225–1232. IEEE, 2016.

[82] Kendall Lowrey, Svetoslav Kolev, Jeremy Dao, Aravind Rajeswaran, and Emanuel Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. *CoRR*, abs/1803.10371, 2018.

[83] Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

[84] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[85] William Montgomery, Anurag Ajay, Chelsea Finn, Pieter Abbeel, and Sergey Levine. Reset-free guided policy search: efficient deep reinforcement learning with stochastic initial states. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3373–3380. IEEE, 2017.

[86] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. Todorov. Interactive Control of Diverse Complex Characters with Neural Networks. In *NIPS*, pages 3132–3140, 2015.

[87] I. Mordatch, K. Lowrey, and E. Todorov. Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *2015 IEEE/RJS International Conference on Intelligent Robots and Systems*, September 2015.

[88] Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 242–248. IEEE, 2016.

[89] Igor Mordatch, Zoran Popović, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144. Eurographics Association, 2012.

[90] Igor Mordatch and Emanuel Todorov. Combining the benefits of function approximation and trajectory optimization. In *RSS*, 2014.

[91] Igor Mordatch, Emanuel Todorov, and Zoran Popovic. Discovery of complex behaviors through contact-invariant optimization. *ACM SIGGRAPH*, 31(4):43, 2012.

[92] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *NIPS*, 2016.

[93] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 2008.

[94] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018.

[95] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[96] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

[97] A.M. Okamura, N. Smaby, and M.R. Cutkosky. An overview of dexterous manipulation. In *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00*, volume 1, pages 255–262 vol.1, 2000.

[98] OpenAI. Learning dexterous in-hand manipulation. *ArXiv e-prints*, abs/1808.00177, 2018.

[99] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *CoRR*, abs/1806.03335, 2018.

[100] Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *ICML*, 2016.

[101] Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pages 3003–3011, 2013.

[102] Yunpeng Pan and Evangelos A Theodorou. Data-driven differential dynamic programming using gaussian processes. In *American Control Conference (ACC), 2015*, pages 4467–4472. IEEE, 2015.

[103] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[104] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, volume abs/1710.06537, pages 1–8. IEEE, 2018.

[105] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2007.

[106] Jan Peters. Machine learning of motor skills for robotics. *PhD Dissertation, University of Southern California*, 2007.

[107] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

[108] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA*, 2016.

[109] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[110] Ph Poignet and M Gautier. Nonlinear model predictive control of a robot manipulator. In *6th International Workshop on Advanced Motion Control. Proceedings (Cat. No. 00TH8494)*, pages 401–406. IEEE, 2000.

[111] Michael Posa, Scott Kuindersma, and Russ Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1366–1373. IEEE, 2016.

[112] Michael Posa and Russ Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic foundations of robotics X*, pages 527–542. Springer, 2013.

[113] Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami. Capture point: A step toward humanoid push recovery. In *2006 6th IEEE-RAS international conference on humanoid robots*, pages 200–207. IEEE, 2006.

[114] S. Joe Qina and Thomas A. Badgwellb. A survey of industrial model predictive control technology. *Control Engineering Practice*, 2003.

[115] Ali Rahimi and Benjamin Recht. Random Features for Large-Scale Kernel Machines. In *NIPS*, 2007.

[116] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

[117] Joose Rajamäki and Perttu Hämäläinen. Augmenting sampling based controllers with machine learning. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pages 11:1–11:9, New York, NY, USA, 2017. ACM.

[118] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. EPOpt: Learning Robust Neural Network Policies Using Model Ensembles. In *ICLR*, 2017.

[119] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *CoRR*, abs/1709.10087, 2017.

[120] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[121] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards Generalization and Simplicity in Continuous Control. In *NIPS*, 2017.

[122] Stéphane Ross and J. Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. In *ICML*, 2012.

[123] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[124] Fereshteh Sadeghi and Sergey Levine. (CAD)2RL: Real Single-Image Flight without a Single Real Image. In *RSS*, 2016.

[125] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *ICML*, 2015.

[126] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[127] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[128] C-L Shih, Yun Zhu, and William A Gruver. Optimization of the biped robot trajectory. In *Conference Proceedings 1991 IEEE International Conference on Systems, Man, and Cybernetics*, pages 899–903. IEEE, 1991.

[129] David H Shim, H Jin Kim, and Shankar Sastry. Decentralized nonlinear model predictive control of multiple flying robots. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 4, pages 3621–3626. IEEE, 2003.

[130] Jennie Si. *Handbook of learning and approximate dynamic programming*, volume 2. John Wiley & Sons, 2004.

[131] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[132] William D Smart and Leslie Pack Kaelbling. Practical reinforcement learning in continuous spaces. In *ICML*, pages 903–910, 2000.

[133] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[134] Benjamin J Stephens. State estimation for force-controlled humanoid balance using simple models in the presence of modeling error. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3994–3999. IEEE, 2011.

[135] Benjamin J Stephens and Christopher G Atkeson. Push recovery by stepping for humanoid robots with force controlled joints. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 52–59. IEEE, 2010.

[136] Tomomichi Sugihara, Yoshihiko Nakamura, and Hirochika Inoue. Real-time humanoid motion generation through zmp manipulation based on inverted pendulum control. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1404–1409. IEEE, 2002.

[137] Wen Sun, J. Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *CoRR*, abs/1805.11240, 2018.

[138] Wen Sun, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Dual policy iteration. *CoRR*, abs/1805.10755, 2018.

[139] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[140] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[141] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *IROS*, pages 4906–4913, 2012.

[142] Yuval Tassa, Nicolas Mansard, and Emanuel Todorov. Control-limited differential dynamic programming. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175, 2014.

[143] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[144] Gerald Tesauro and Gregory R Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997.

[145] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11:3137–3181, 2010.

[146] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[147] Sebastian B Thrun. Efficient exploration in reinforcement learning. 1992.

[148] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017.

[149] Emanuel Todorov. Linearly-solvable markov decision problems. In *NIPS*, 2006.

[150] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[151] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *ACC*, 2005.

[152] Pramod Vachhani, Shankar Narasimhan, and Raghunathan Rengaswamy. Robust and reliable estimation via Unscented Recursive Nonlinear Dynamic Data Reconciliation. *Journal of Process Control*, 16(10):1075–1086, December 2006.

[153] Jack M Wang, David J Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, volume 29, page 73. ACM, 2010.

[154] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, March 2010.

[155] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1433–1440. IEEE, 2016.

[156] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[157] Y. Wu, D. Hu, M. Wu, and X. Hu. Unscented kalman filtering for additive noise case: Augmented versus nonaugmented. *IEEE Signal Processing Letters*, 12(5):357–360, May 2006.

[158] X Xinjilefu, Siyuan Feng, Weiwei Huang, and Christopher G Atkeson. Decoupled state estimation for humanoids using full-body dynamics. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 195–201. IEEE, 2014.

[159] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *arXiv preprint arXiv:1610.00673*, 2016.

[160] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 100–107. IEEE, 2013.