The Expert is the Obstacle
Building a General Framework for Learned Robot Motion

Adam Harper Fishman

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

Doctor of Philosophy

University of Washington
2024

Reading Committee:
Dieter Fox, Chair
Byron Boots, Co-Chair
Abhishek Gupta

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

The Expert is the Obstacle

Building a General Framework for Learned Robot Motion

Adam Harper Fishman

Chair of the Supervisory Committee:
Dieter Fox
Computer Science & Engineering

Co-Chair of the Supervisory Committee:
Byron Boots
Computer Science & Engineering

There are many ways to move a robotic arm through space, but each technique comes with assumptions and trade-offs. Some techniques provide quick, local solutions, while others provide theoretical feasibility guarantees. Furthermore, most state-of-the-art techniques rely on a precomputed scene model to ensure safety. In highly dynamic environments, such as when a robot must operate in a fast-paced industrial setting or around human partners, this assumptions may break down as obstacles move in and out of view. Humans are able to operate effortlessly in these settings, relying on our vast experience to make quick decisions, even under uncertainty.

In recent years, we have seen an incredible proliferation of empirical machine learning approaches to long-standing problems, ranging from solving challenging games to producing human-like language. My PhD research has focused on adapting these ideas to robot motion in order to balance the trade-offs of traditional algorithmic approaches to motion generation.

In this dissertation, I describe my PhD research on motion generation techniques for robotic manipulation. I will cover my work on trajectory optimization applied to dynamic, multi-agents settings and my research on using large scale imitation learning to create a safe end-to-end policy for manipulator control.

To my grandfather, Ernest Stern,
who taught me the importance of science,
and the joy of research with
dirt under my fingernails

ACKNOWLEDGEMENTS

Firstly, I'd like to thank my partner Jenny Mayer. Through the high-highs and low-lows of my PhD, you have been a fount of joy and a steadying source of moral support. I'd also like to thank my grandmother Elaine Fishman, my grandfather Ernie Stern, my parents Sara Stern and Ted Fishman, my sister Elly Fishman, and my niece Zuzie Gaster. Without your love and guidance, the work described in this document would not have been possible. I'd also like to thank the academics in my family, my aunt Jessie Stern and my brother-in-law Jonah Gaster, for showing me the ropes.

To my friends Paul Cathey, Isabel Strauss, Will Craft, and Jeff Bruchez, thank you for always providing highly biased, often unwise, advice whenever I needed it. To my friends Scott McKinney and Ben Segal, thank you for teasing me so hard about leaving California to get a PhD that I ended up doing it. To my friends Sharif Vakili, Lorenzo Labitigan, and Greg Hindy, thank you for showing me that no matter how busy I am, there is always time to care for the people I love. To my friend Erich Owens, thank you for introducing me to computer vision–you sent me a PDF and it changed my life.

To my mathematical mentors PJ Karafiol and Douglas O'Roarke, thank you for inspiring me to love math, especially when it's hard. To Paul J. Sally Jr., thank you for teaching me that an academic's role is as much service to their community as it is the relentless pursuit of knowledge.

One main goal of my PhD was to learn good taste in problems and I am immeasurably grateful to my advisors Dieter Fox and Byron Boots for guiding me toward an understanding of what problems are meaningful, how to communicate their worth, and what to do when nothing works. There were many times over these past six years when, deep in a hard problem, I felt lost at sea and unable to see the shore.

I am thankful for my advisors' patience and encouragement as they told me to stay focused and keep swimming.

To my committee members Abhishek Gupta and Karen Leung, thank you both for your constructive feedback, career advice, and challenging questions. To my former advisor Steve Seitz, thank you for admitting me to the program despite my unusual background, for inspiring me to constantly look for new, unexplored problems, and for supporting my shift from 3D computer vision into robotics. To the CSE department staff, especially Elise deGoede Dorough, thank you for helping me navigate a PhD with many twists and turns.

To my collaborators Adithya Murali and Clemens Eppner, thank you for your patience, guidance, and wisdom while we toiled away at MπNets. I was completely stuck before you both offered to help and without you on the project, for all we know, I might still be stuck now. To Bala Sundaralingam, thank you for your tireless efforts to *make things work* and for your unabashed honesty in what must be fixed. To Wei Yang, thank you for helping me find my footing in research. To Chris Paxton, thank you for advising me on what's possible and teaching me what's hard in both deep learning and motion planning. To Nathan Ratliff, thank you for your patience and pedagogy as I learned about motion planning, trajectory optimization, policy learning, and control. Perhaps most importantly, you opened my eyes to the idea of understanding math intuitively, an idea that seemed alien at the time but now feels like water to a fish. To Ryan Julian, thank you for your mentorship, support, and friendship as I learned a wholly new research style. And, thank you for helping me understand the complex landscape of how research intersects with industrial motivations.

To Aaron Walsman and Zoey Chen, thank you for your technical aid and emotional support through these last six years. To Chris Xie and Daniel Gordon, thank you for the many long back-and-forths as I tried to distill your brilliance and experience into unit tests for neural networks. To Mohak Bhardwaj and Wentao Yuan, thank you for your help on *Avoid Everything*—you each helped improve the system in ways that would have been immensely challenging me for me to

do alone. To Simran Malhi and Nikhil Sridhar, thank you for your diligence and curiosity as we explored new ideas together. To Tanner Schmidt and Arun Byravan, thank you for giving me a north star to follow and advice of how to get there. To Keunhong Park, Aleks Holynski, Xuan Luo, and JJ Park, thank you for welcoming me into GRAIL and helping me find my way through graduate school. My time in the lab was short, but I hope our friendships can live on forever. To Schmittle and Rosario Scalise–the brain trust–thank you for answering my many (many) stupid questions about motion planning and for advising me against accidentally jumping headfirst off a cliff with some of my bolder statements on the field. To Selest Nashef, thank you for your inextinguishable positivity, even when all the computers are broken and everyone is panicking.

Lastly, I'd like to thank my dog Sequoia for showing me every day how to keep my eye on the ball.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

The landscape of robotics is quickly changing. Following the success of neural networks in related fields, similar machine learning techniques are quickly becoming the de facto standard for many parts of the robotics stack. While robotic systems have traditionally separated perception and control [171], recent advances in deep learning have led to a rise in end-to-end policies for robotic control. Unlike traditional analytic techniques, neural networks are difficult to understand, as they are trained empirically rather than following a logical sequence of steps. Algorithmic intelligibility has been especially important for robotics as it has enabled us to trust that our systems will behave as expected, assuming that the overall system satisfies the algorithm's requirements. These requirements can be severely limiting, especially as we work to deploy robots in highly complex, dynamic environments that are not easily modeled. Although analytic techniques provide safety guarantees, their strict input specifications often cause these guarantees to break down in real-world settings, resulting in unexpected failures. It is exactly in these underspecified settings where learning-based methods excel due to their ability to estimate through experience. Yet, these methods can also fail unexpectedly, particularly in settings not well-represented in the training data. Despite these challenges, there is a clear need to create learned robotic systems that can perform safely through the complexity of the real world. In order to satisfy both safety requirements and dynamic capabilities, our work has spearheaded the use of learned systems in place of analytic methods to create robust systems with strong, safe performance. While my PhD research points strongly in the direction of learned motion policies, these systems rely on analytic methods for guidance via expert demonstration and focused correction of learned behavior. Through the combination of these ideas—end-to-end learn-

ing and analytic planning—our work inches the field closer to a safe, generalizable model for robot motion.

To understand the power of learned methods for robotics, it is critical to understand the limitations of classical systems. While there are many ways to move a robotic arm through space, each technique comes with significant trade-offs that impact feasibility of use in the real world. Classical planning techniques [29, 58, 73, 85, 86, 92], may guarantee solutions, but they are often slow and require configuration space (joint angle) goals, an unintuitive representation for common tasks defined in 3D space. Classical local control and MPC methods [14, 24, 173] are typically fast and allow for intuitive task-space targets, but they provide no guarantee of a solution. Trajectory optimization [42, 48, 134, 146, 157] may produce a locally optimal path, but requires high expertise to tune cost functions for specific tasks. Finally, all of these techniques require a complete scene representation to promote safe behavior, which can be difficult or impossible to acquire in dynamic settings.

Meanwhile, human motion does not require these trade-offs. We operate in partially observed scenes, finding little challenge in interacting with new objects. If and when humans collide with the environment, we easily recover, often without damage done to ourselves or the environment. For most humans, moving our arms is an afterthought, and yet robots still require complex systems to produce a poor imitation of human movement.

In recent years, we have seen immense success of large scale machine learning systems for many complex and under-specified tasks, ranging from language generation [1] to coding [55] to video generation [20]. The problem of robot motion generation has many of these same challenges—the goals may be vague, the potential paths may be highly multi-modal, and the solutions must be highly precise to prevent catastrophic system failure.

In my research, I have focused on using machine learning to balance the trade-offs of different methods of motion generation to more closely mimic natural human motion. Machine learning methods are excellent at reasoning about 3D scenes in partially observed settings;

overparametrized neural networks are well suited to produce complex behavior in unseen environments. Machine learning methods do not provide the same guarantees as traditional techniques, but my research has demonstrated that they are often superior in terms of their safety, reactivity, and ability to reach set targets, while also requiring few assumptions to be made about the problems and environments.

In this dissertation, I describe my PhD research on motion generation techniques for robotic manipulation and how we can leverage large datasets and deep learning to achieve performance with learned policies.

In Part I, I present an optimization framework for multi-agent teamwork. In this work, we formulate the idea of a collaborative interaction model to describe the way many agents collaborate. Then, we describe the concept of a collaborative behavior model, which is the optimization problem each agent must solve independently in order to achieve ideal system performance. To account from deviations from the optimal joint behavior—agents without shared knowledge do not collaborate optimally—we continually re-optimize the robot's behavior in an MPC loop. Finally, we demonstrate this framework's efficacy on the real world task of human robot handover.

In Part II, I present our work on large scale imitation learning for end-to-end motion manipulator control. In this work, we presented what was, at-the-time, the largest foundation model effort for manipulator planning and control. This model, which we call Motion Policy Networks (MπNets), was trained on a dataset of millions of expert demonstrations and outperforms the prior state-of-the-art neural motion planner by 46%. MπNets works well in partially observed environments and for navigating through complex tabletop settings. Compared to baseline methods, MπNets work demonstrated significantly stronger performance end-to-end motion in novel settings. Despite its benefits, this work cannot replace analytic techniques, and in this section, I will thoroughly discuss the comparative performance against traditional techniques.

In Part III, I present an advancement in our large scale imitation work that expands the types of problems it can solve, while signif-

icantly improving the safety. This work, which we titled *Avoid Everything*, consists of a policy architecture that is better able to maintain the scene's 3D structure and a novel fine-tuning algorithm that uses optimization to locally correct trajectories predicted by a pre-trained policy. With these techniques combined, we tested among an randomly constructed set of problems in challenging, partially settings and were able to solve 63% of those where MπNets failed.

Finally, I will present some ongoing and future work that explores how popular techniques in the contemporary imitation learning literature apply to the problem of end-to-end collision avoidance.

Part I

TRAJECTORY OPTIMIZATION

In this chapter, I detail my research on multi-agent trajectory optimization and how it can be applied to the problem of human-robot teamwork. Human-robot collaboration is a challenging problem because the robot cannot know the human's true intent. In [48], we detailed an MPC-style approach to trajectory optimization, where the system continuously re-optimizes based on new state information. While this work describes an optimization-based planner, it also demonstrates how inference and planning are inseparable in highly dynamic settings. Within the paradigm proposed, the optimizer is simultaneously planning a trajectory while inferring the human's optimal future behavior. While this work demonstrated the flexibility of optimization for complex multi-agent problems, it also highlighted many of the challenges of using optimization for these settings. When it was properly tuned, the system exhibited superior performance over other methods, but we found that the system had to be constantly retuned in order to achieve strong results. Furthermore, our cost function for the human was very simple and would be very challenging to scale to more complex tasks without using something such as inverse reinforcement learning [184] to learn an appropriate set of cost functions.

2

COLLABORATIVE INTERACTION MODELS FOR OPTIMIZED HUMAN-ROBOT TEAMWORK

2.1 INTRODUCTION

Human behavior is determined by a mixture of intent, world prediction, anticipation, physical limitations, and more. When planning in the presence of people, robotic decision processes often encapsulate these diverse desiderata under the lid of a black box dynamics function. When the robot and human's goals are independent [3, 96, 186], this model has been very successful.

However, cooperating to achieve shared goals is more difficult. Take the human-robot hand-over task shown in Fig. 1 as an example, where a robot must receive some object from a human collaborator. Humans will act based on what they imagine the robot will do [43], and, conversely, the robot should choose actions based on its best estimate of the human's intention. Predicting human intention while planning is not new, this has been explored in anticipatory planning [80], and prior work has modeled human kinematics and dynamics in order to achieve collaborative manipulation tasks [120, 165, 170].

However, humans often do not act according to plan. Any robot planner that tries to predict their intentions must be highly reactive. We propose a Model Predictive Control (MPC) approach, which models both human and robot as separate, fully-actuated actors in a combined trajectory optimization problem. Our MPC approach allows the robot to determine the most effective form of collaboration while still being able to react to changing circumstances and noisy sensor data. We specifically apply our approach to the problem of human-robot handover [64, 95, 98, 152]. The core problem is that the human and robot must coordinate on where and how the handover is to take place [152]. In effect, we must balance between the two cost functions

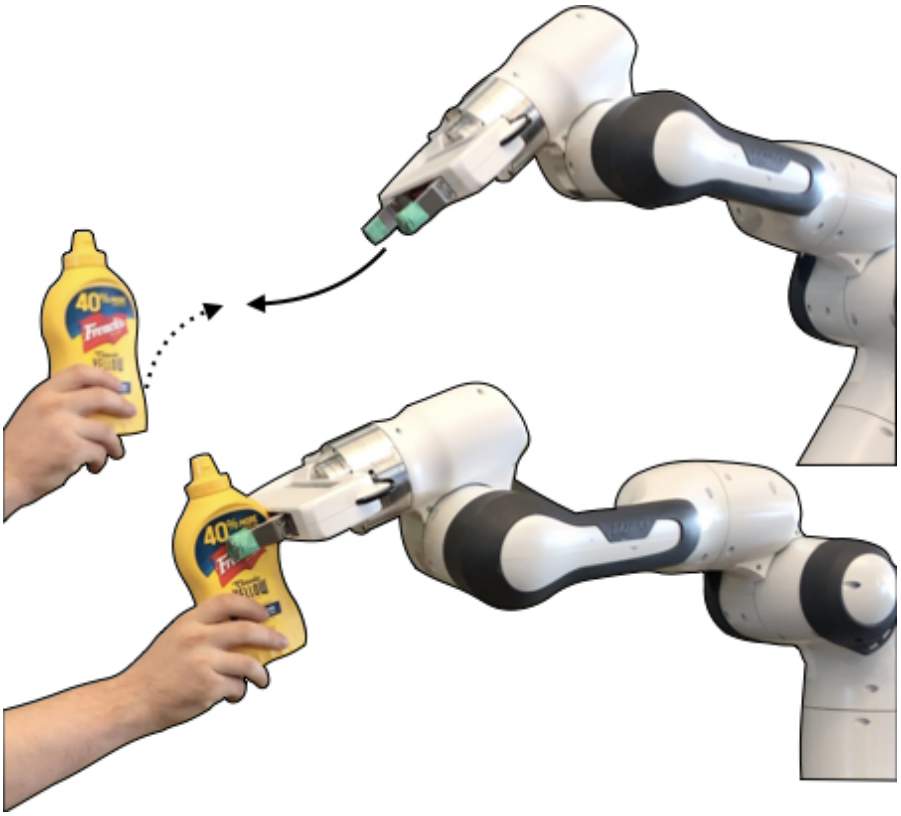


Figure 1: Coordinating a fluid human-robot handover requires an estimate of the human's plan, so that the robot can be in position to make the handover at the correct time. Our algorithm can achieve smooth and natural human-robot collaborative motions in a variety of scenarios, even in the presence of obstacles and sensor uncertainty.

for human and robot, avoiding obstacles while finding the most logical location for both to reach.

This combined human-robot system is both partially observable and under-actuated since the robot has no real control over the human and cannot directly observe factors influencing their decisions. Therefore, at a minimum, our planner must be *reactive* [110] to unforeseen human behavior. We follow a real-time Model-Predictive Control (MPC) paradigm and re-optimize with each new observation. Computation speed is also crucial. We employ a modern motion optimization strategy, which leverages fast Gauss-Newton solvers [105, 133, 164], and assume relevant aspects of the human and robot are fully actuated.

Additionally, to ensure spatial consistency of the resulting reactive behavior through re-optimization, we introduce a novel class

of explicit sparse reward terms, *i.e.*, negative costs, around the target. Within a certain radius, the robot is explicitly rewarded for approaching the target, thus extending the target's influence beyond a terminal potential to each intermediate time step. The system is therefore able to compromise between goal accumulation and trajectory smoothness.

We evaluate our technique in both simulated ablation studies as well as real-world handovers between a human participant and a Franka robot using a real-time perception system. We show that, especially in the presence of obstacles, our technique enables the robot to anticipate the human's actions leading to well-coordinated, quick, and smooth handover behavior while timing the handover better than the alternatives, both quantitatively and qualitatively.

2.2 RELATED WORK

Modeling human behavior is crucial for successful human-robot collaborative manipulation and has been explored in a variety of contexts [3, 95, 165, 182]. In addition, many recent methods for human-robot handover use perception and some manner of human modeling to achieve reactivity [84, 95, 120]. However, these models are usually uni-directional, with information flowing from prediction to planner but not vice versa. For example, Ziebart et al.[186] used predicted goal-oriented pedestrian behavior to augment navigation planners to minimize interaction. Similarly, Mainprice et al. [96] modeled human reaching behaviors to reduce interaction or collision events while working along-side humans. Maeda et al. [95] use probabilistic motion primitives to model both humans and robots in a variety of collaborative tasks, including handover. Zhou et al. [182] used a recurrent neural net to model human activity for collaboration in the operating room.

In reality, however, the human will respond to qualities of the robot's motion, *e.g.*, speed and shape, trying to estimate and adapt to its motion. Humans and robots can collaborate more effectively if the robot's motions appear legible to humans [43] and allow the humans

to understand the robot's goals [57]. One way to achieve legibility is to use human demonstration data to teach the robot [95]. Another approach focuses on jointly modeling the human-robot system as some sort of hybrid planning problem, [3, 151, 165], and try to structure the problem to ensure effective collaboration.

Human-robot handovers are a particularly well-studied area for human-robot collaboration, with applications both to industry [120, 167] and to in-home assistance [64]. Much prior work analyzes the formulation of the human handover and how to structure the action naturally [64, 152]. This can be particularly well represented as a hybrid planning problem [151, 165]. Toussaint et al. [165] proposed a method for offline planning based on Task and Motion Planning. This allows for longer-horizon planning across grasps as compared to our method, but is inherently less reactive. Other work used a dyadic model for collaboration between a human and a robot [151].

Our method applies more specifically to the approach phase of the handover. Related ideas include exploiting a database of human demonstrations to produce natural and fluid plans [174]. Likewise, Maeda et al. [95] use imitation learning to mimic human behavior and Medina et al. [98] use a human-inspired dynamics controller to model the entire action: approach, grasp, retract. These methods are well-suited for controlled interaction settings, but generalizing them to handle the diversity of speed and environmental variations encountered in the real world is challenging. Peternel et al. [120] also model the human during collaborative manipulation, but their goal is to minimize risk of injury, whereas our goal is to achieve fluid collaboration in the presence of obstacles.

Our work relies on motion optimization approaches that are both fast and expressive [105, 133, 164]. Motion planning as an optimization problem was first presented in [135], and accelerated in a quick progression of work [28, 71, 147]. These early optimizers addressed primarily the subproblem of smooth collision avoidance. The work of [133, 162, 163], extended the paradigm showing that generic second-order Gauss-Newton optimizers out-of-the-box could solve a more general class of constrained motion optimization problem. Soon there-

after, Mukadam et al. [105] demonstrated that standard factor graph tools could drastically simplify the modeling. We build on these ideas here, using a factor graph to model the problem and fast modern optimizers to solve the continuous optimization loop in real time.

While our setting is fundamentally partially observable, we do not address the Partially Observable Markov Decision Process (POMDP) problem directly, other than to use standard, reactive maximum a posteriori (MAP) approximation techniques [110] to motivate the importance of continuous re-optimization. Using maximum-likelihood observations and active replanning has proven useful before, even for very complex multi-stage tasks [54]. Other approaches use Monte Carlo sampling to explore possible outcomes for various actions [3, 149, 150]. Some POMDP work has even actively modeled uncertainty over human intention [3, 7], particularly in the context of autonomous vehicles [3].

2.3 THEORETICAL FRAMEWORK

In games, agents try to optimize individual objectives [160], but collaborative tasks require cooperation. When collaborating, agents collectively optimize a single system objective. In this section, we formalize the collaborative system. We derive predictive models for each external, *i.e.*, uncontrolled, agent and an optimal control objective for the controlled agent, *i.e.*the robot. We then show that if the models of the external agents' behavior are sufficiently predictive, the controlled agent can achieve a stable *collaborative equilibrium* by choosing actions according to its objective.

We consider a system constituting $N + 1$ total agents. Let the 0^{th} agent denote the controlled agent and agents $1, \ldots, N$ be external, uncontrolled, but collaborating agents.

2.3.1 *Collaborative interaction model*

Denote the i^{th} agent's trajectory by $\boldsymbol{\xi}_i = (\mathbf{q}_i^0, \mathbf{q}_i^1, \cdots, \mathbf{q}_i^{T+1})$. Let $\boldsymbol{\sigma}_i^t = (\mathbf{q}_i^{t-1}, \mathbf{q}_i^t, \mathbf{q}_i^{t+1})$ denote the trajectory's t^{th} second-order clique,[1] a triple of consecutive positions used to represent position and the corresponding finite-difference approximations of velocity, and acceleration at each time step [163].

We define the joint *collaborative system trajectory* as $\boldsymbol{\xi} = (\boldsymbol{\xi}_0, \boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_N)$ and denote its constituent 2nd-order cliques by $\boldsymbol{\sigma}^t = (\boldsymbol{\sigma}_0^t, \boldsymbol{\sigma}_1^t, \ldots, \boldsymbol{\sigma}_N^t)$.

Denoting the space of all collaborative system trajectories by Ξ, we define the system's *collaborative interaction model* (or simply its *collaboration model*) $\mathcal{M} = \{C, G, H\}$ as

$$\min_{\boldsymbol{\xi}} C(\boldsymbol{\xi}) \quad \text{s.t.} \quad G(\boldsymbol{\xi}) \leqslant 0,\ H(\boldsymbol{\xi}) = 0 \tag{1}$$

where $C : \Xi \to \mathbb{R}$ is the collaborative cost, $G : \Xi \to \mathbb{R}^k$ are k inequality constraint functions, and $H : \Xi \to \mathbb{R}^l$ are l equality constraint functions. For compactness, we use $\Xi_{\mathcal{M}} \subset \Xi$ to denote the feasible set of trajectories that satisfy the constraints G and H. We can then write the collaborative interaction model as $\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi} \in \Xi_{\mathcal{M}}} C(\boldsymbol{\xi})$. The model for a given collaborative system can change incrementally over time as the environment, the agents' goals, or the agents themselves change. We assume that the optimizer is able to track solutions over time within a continuous optimization loop, such as Model Predictive Control (MPC). Note that both the costs / constraints and the set of available trajectories $\Xi_{\mathcal{M}}$ usually change from cycle to cycle updated with the latest estimates of the world and agent states.

In our experiments, we export a k^{th}-order Markov structure in the system [163] enabling us to write the collaborative interaction model of Equation 1 in clique notation as

$$\min_{\xi} \sum_{t=1}^{T} c_t(\boldsymbol{\sigma}^t) \ \text{ s.t. } \ g_t(\boldsymbol{\sigma}^t) \leqslant 0,\ h_t(\boldsymbol{\sigma}^t) = 0 \ \ \forall t. \tag{2}$$

1 Here we use superscripts just for notational convenience of time indexing, not to be confused with the component indexing of tensor notation.

Often, more complex task spaces are defined on these cliques by transforming them through differentiable task maps where objective terms may reside. It is common to represent the task spaces, *i.e.*, the codomains of the task maps, with maximal coordinates, which are an explicit representation of the task space constrained to match the output of the task map. Following this paradigm, we define our optimization costs and constraints in maximal coordinates on the relevant task space. We also use soft constraints implemented as unconstrained penalties in our experiments as in [38]. In this section, though, we use the more compact notation given above for succinctness and generality.

The key intuition behind our model is that although we cannot explicitly control the N external collaborating agents, we assume we can sufficiently predict their behavior and treat prediction errors as system disturbances. We make this assumption concrete below and explore it experimentally in Sec. 2.4.2.

When the collaborative interaction model has a unique global minimum, that minimizer acts as an equilibrium point and becomes predictable by the agents in a way we can exploit in our model (explored below in Section 2.3.3). We, therefore, call the global minimum the system's *collaborative equilibrium* and say the system is well-defined if it has a unique global minimum. In this work, we assume both that the global minimum is well defined and that an optimizer will be able to track the global minimum over time. In practice, these assumptions amount to the agents mutually knowing the higher-level collaboration plan either in advance or by sufficiently communicating it to each other unambiguously on the fly. For complex tasks, there may be many local minima or even regions of equally good global minima, representing different equilibria. In these cases, the system would require additional estimation machinery to maintain predictive distributions across external agent behavior, which we do not address here.

We start by defining explicitly the agents' individual predictive *collaborative behavior models* implicit in the above collaborative interaction model. Let $\Xi_{\mathcal{M}}[\boldsymbol{\xi}_i]$ denote the feasible set of system trajectories where

the i^{th} agent's trajectory is fixed at $\boldsymbol{\xi}_i$. We define the i^{th} agent's *predictive collaborative cost* to be

$$c_i(\boldsymbol{\xi}_i) = \min_{\boldsymbol{\xi}^{\setminus i} \in \Xi_{\mathcal{M}}[\boldsymbol{\xi}_i]} C(\boldsymbol{\xi}_i, \boldsymbol{\xi}^{\setminus i}), \tag{3}$$

where with a slight abuse of notation, we use $C(\boldsymbol{\xi}_i, \boldsymbol{\xi}^{\setminus i})$ to denote the collaborative cost evaluated at the joint system trajectory defined by agent i's trajectory $\boldsymbol{\xi}_i$ and the remaining system trajectories $\boldsymbol{\xi}^{\setminus i}$ of all other agents $j \neq i$. This cost encodes the agent's action criteria under an assumption that all other agents are predicted as having optimal collaborative responses under the system's collaboration model. Note that these predictive models are assumed to know agent i's intent (the trajectory $\boldsymbol{\xi}_i$). While this assumption is generally wrong, as the robot cannot truly know an external agent's intent, we will see below that it is valid at the system's collaborative equilibrium where equilibrium behavior becomes mutually predictable (see Section 2.3.3).

Each agent then has its own individual *collaborative behavior model* of the form

$$\boldsymbol{\xi}_i^* = \arg\min_{\boldsymbol{\xi}_i \in \Xi_M^i} c_i(\boldsymbol{\xi}_i), \tag{4}$$

where Ξ_M^i is the set of all trajectories $\boldsymbol{\xi}_i$ for the i^{th} agent for which $\Xi_M[\boldsymbol{\xi}_i]$ is nonempty, *i.e.*, $\Xi_M^i = \{\boldsymbol{\xi}_i \mid \Xi_M[\boldsymbol{\xi}_i] \neq \emptyset\}$

2.3.2 *Stability of the predictive controller*

We adopt definitions of stability from control theory and say that an assignment of behavior generation algorithms to the agents are collectively, or asymptotically, stable around the equilibrium if the joint system evolves stably, or stably asymptotically, around the system trajectory. Under this notion of stability, we can make following statement.

Lemma 1. *Suppose we have a collaborative system and a corresponding collaboration model $\mathcal{M}$. If we can say that a MPC algorithm over $\mathcal{M}$ rejects ϵ-disturbances and that each agent's collaborative behavior model is*

ϵ-predictive of the agent's next action, including the controlled agent's execution under the environment's stochasticity, then controlling the controlled agent with the MPC algorithm will create system behavior that is stable around the collaborative equilibrium of $\mathcal{M}$.

In other words, if our collaboration model is sufficiently predictive for the external agents and we control our controllable agent using the collaborative behavior model derived from it, the combined system behavior is stable around the collaborative equilibrium.

Note that in this stability statement, the metric ϵ-predictive is undefined. This is because the statement will hold as long as the definition of ϵ-predictive is consistent with the definition of ϵ-disturbances, *i.e.*, the range of system deviations that can be handled by MPC. While we cannot explicitly control the external agents, if we can predict their behavior sufficiently well, then we may treat deviations as system disturbances. With this, we do not need to assume that the external agents generate behavior with the same collaborative system model, as the collaborative behavior models induced by the system are sufficiently predictive.

2.3.3 *Mutual predictability of the collaborative equilibrium*

Each collaborative behavior model implicitly uses a *conditional* model to predict the behavior of external agents. Specifically, under agent i's collaborative behavior model, the cost $c_i(\boldsymbol{\xi}_i)$ optimizes over each external agent $j \neq i$ *given* agent i's trajectory $\boldsymbol{\xi}_i$, thus modeling the response the other agents would have if they were given knowledge of $\boldsymbol{\xi}_i$. The model assumes that all responding agents know the agent i's intent, which is in general not true. However, equilibrium behavior has a mutual predictability property which enables all agents behavior to be predictable, thereby validating the conditional model specifically at the collaborative equilibrium.

Equilibrium predictions of other agent's behavior are both reflexive and transitive, creating a stationarity property of the predictions. For instance, let $\boldsymbol{\xi}_j^{\mathcal{M}}(\boldsymbol{\xi}_i)$ be the implicit prediction made by agent i of

how agent j will respond to agent i's intended trajectory $\boldsymbol{\xi}_i$. Let $\boldsymbol{\xi}^* = \{\boldsymbol{\xi}_i^*\}_{i=1}^N$ denote the collaborative equilibrium of system $\mathcal{M}$. Then for all i, j we have $\boldsymbol{\xi}_j^{\mathcal{M}}(\boldsymbol{\xi}_i^*) = \boldsymbol{\xi}_j^*$. Therefore, $\boldsymbol{\xi}_i^{\mathcal{M}}(\boldsymbol{\xi}_j^{\mathcal{M}}(\boldsymbol{\xi}_i^*)) = \boldsymbol{\xi}_i^*$ (reflexive) and $\boldsymbol{\xi}_k^{\mathcal{M}}(\boldsymbol{\xi}_j^{\mathcal{M}}(\boldsymbol{\xi}_i^*)) = \boldsymbol{\xi}_k^* = \boldsymbol{\xi}_k^{\mathcal{M}}(\boldsymbol{\xi}_i^*)$ (transitive).

In other words, each agent predicts an equilibrium response under equilibrium behavior. Even though the agent uses a conditional predictive model which assumes external agents know the agent's intent, specifically at the collaborative equilibrium, the agent's intended behavior becomes predictable as part of the equilibrium behavior validating the use of the conditional model.

2.4 HUMAN-ROBOT HANDOVER USING FINITE-HORIZON OPTIMIZATION

In this section, we formulate the handover task as an application of our general framework where the collaborative model optimizes for the human and robot successfully reaching each other to perform the handover. We detail the objective terms used in our models (Sections 2.4.1, 2.4.2, 2.4.3) and discuss implementing spatially consistent behavior using finite-time-horizon MPC (see Section 2.4.4).

In this section, we consider the robot to be the controlled agent (agent 0) and the human to be the uncontrolled external agent (agent 1), and denote their trajectories as $\boldsymbol{\xi}^R = \{\mathbf{q}_i^R\}_{i=0}^T$ (robot) and $\boldsymbol{\xi}^A = \{\mathbf{q}_i^A\}_{i=0}^T$ (external agent, *i.e.*human), respectively. We focus here on defining the unconstrained objective, making the common assumption that many constraints can be naturally modeled as soft constraints using fixed penalties (see, for instance, [38]). This is a reasonable approximation, especially since stochasticity makes the optimization inherently approximate.

The collaboration objective can be decomposed into three terms, a robot specific term, an external agent (human) specific term, and an interaction term.

$$C(\xi) = \lambda_R c^R(\boldsymbol{\xi}^R) + \lambda_A c^A(\boldsymbol{\xi}^A) + \lambda_I c^I(\boldsymbol{\xi}^R, \boldsymbol{\xi}^A) \tag{5}$$

We detail these three terms in the following sections.

2.4.1 *Modeling the robot*

First, we define the cost modeling the robot trajectory,

$$c^R(\boldsymbol{\xi}^R) = \sum_{i=0}^{T} c^R(\mathbf{q}_i^R, \dot{\mathbf{q}}_i^R, \ddot{\mathbf{q}}_i^R), \tag{6}$$

where T is the total number of time steps and $\mathbf{q}^R$, $\dot{\mathbf{q}}^R$, $\ddot{\mathbf{q}}^R$ are the position, velocity, and acceleration of the joints of the robot in configuration space. In what follows, we will also use $\mathbf{x}_i^R = \phi(\mathbf{q}_i^R) = [\mathbf{R}_i^R, \mathbf{t}_i^R]$ to represent the 6-DOF pose of the end effector in the world frame, after applying the forward kinematics function $\phi(\cdot)$.

Equation 6 can be split into the sum of individual cost functions, which we define in the following sections.

Obstacle avoidance and joint constraints. To prevent hitting the joint limits and to avoid obstacles, we include three cost functions $c_{joint}(\mathbf{q}_i^R)$, $c_{joint}(\dot{\mathbf{q}}_i^R)$, and $c_{obs}(\mathbf{q}_i^R)$.

Let J denote the indices of the joints, θ_j^R denote the angle of jth joint, and $(\theta_{j,\,min}^R, \theta_{j,\,max}^R)$ denote the corresponding joint limitation, we employ a hinge-loss-based cost [106] for the joint limit:

$$c_{joint}(\mathbf{q}_i^R) \;\; = \sum_{j \in J} \left\| c(\theta_j^R) \right\|^2, \tag{7}$$

where $c(\theta_j^R)$ is defined as

$$c(\theta_j^R) \;\; = \begin{cases} -\theta_j^R + \theta_{j,\,min}^R - \epsilon_j, & \text{if } \theta_j^R < \theta_{j,\,min}^R + \epsilon_j \\ \theta_j^R - \theta_{j,\,max}^R + \epsilon_j, & \text{if } \theta_j^R > \theta_{j,\,max}^R - \epsilon_j \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Here ϵ_j is the joint limit error tolerance for joint j.

We also impose a cost

$$c_{joint}(\dot{\mathbf{q}}_i^R) \;\; = \sum_{j \in J} \left\| c(\dot{\theta}_j^R) \right\|^2, \tag{9}$$

where $c(\dot{\theta}_i)$ is formulated similarly to Equation 8 using $\dot{\theta}$ in place of θ.

These cost functions assume that the environment is static, *i.e.*, the camera and the obstacles are unchanging. Within the context of MPC, however, we are able to update these cost functions as the environment changes and we redefine our optimization problem. We compute a signed distance field representing a discretization of the environment. Then, as in [135], we use a sphere-based "skeleton" that covers the robot's entire volume and surface area. The spheres allow for a sparse and efficient representation of the robot's volume. Our total obstacle cost is then the sum of the cost at each sphere:

$$c_{obs}(\mathbf{q}_i^R) = \sum_{s \in \text{spheres}} \|c(s)\|^2, \tag{10}$$

where

$$c(s) = \begin{cases} -d_s + s_{radius}, & \text{if } d_s \leqslant s_{radius} \\ 0, & \text{if } d_s > s_{radius} \end{cases}. \tag{11}$$

Here d_s is the value of the signed distance function at the sphere s's center.

Velocity and acceleration constraints. We include independent constraints for configuration-space velocity and acceleration constraints, $c(\dot{\mathbf{q}}^R)$ and $c(\ddot{\mathbf{q}}^R)$, that each constrain these values to be zero. The velocity penalty ensures that the robot slows after after reaching its goal, while the acceleration penalty ensures the robot moves fluidly without overshooting its target.

End effector constraints. We also constrain the robot's end effector orientation to match an optimal value and add this to our overall cost function. In the 2D case, this optimal value is straightforward: the robot should always be oriented towards the human. The optimal 3D orientation is more complex.

Let G denote the coordinate frame of the gripper where the z-axis $\mathbf{z}_G$ points directly out from the gripper and the x-axis $\mathbf{x}_G$ points down perpendicular to the gripper. When the gripper is perfectly flat, $\mathbf{x}_G$

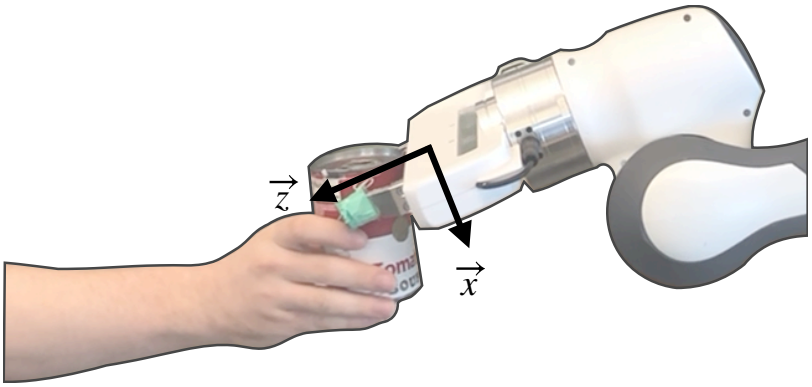


Figure 2: The ideal orientation for the end effector. The z-axis points toward the human's wrist, which we are able to track with the Microsoft Azure Kinect, and the x-axis is as vertical as possible.

points straight down to the ground. We wish to align $\boldsymbol{z}_G$ with $\boldsymbol{v}$, the ray from the end effector to the human's hand. We also want the end effector to be approximately flat. We show this ideal configuration in Figure 2. Assuming the world frame has $\boldsymbol{z}_W$ up, we want to find $\boldsymbol{x}_G$ that when expressed in the world coordinates, has the lowest $\boldsymbol{z}$ coordinate, *i.e.*, in the world frame,

$$0 = \boldsymbol{z}_G \cdot \boldsymbol{x}_G = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} \cdot [x_{\boldsymbol{x}_G}, y_{\boldsymbol{x}_G}, z_{\boldsymbol{x}_G}], \tag{12}$$

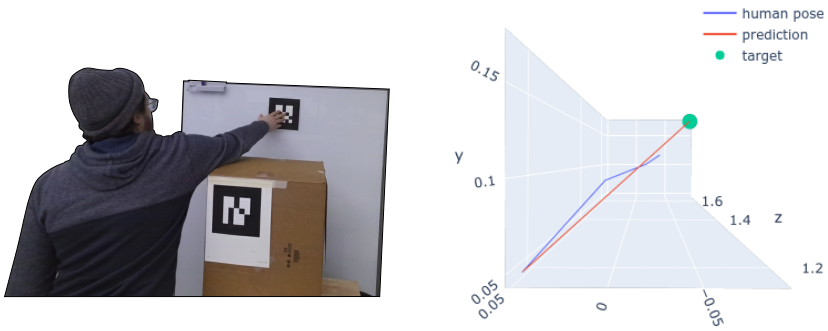
where $[x_{\boldsymbol{x}_G}, y_{\boldsymbol{x}_G}, z_{\boldsymbol{x}_G}]$ correspond to the world frame coordinates of $[1,0,0]_G$ in the gripper frame. We also know $x_{\boldsymbol{x}_G} = 1 - \sqrt{y_{\boldsymbol{x}_G}^2, z_{\boldsymbol{x}_G}^2}$.

If the gripper does not point straight up, we can first solve for $z_{\boldsymbol{x}_G}$, then take the derivative with respect to $y_{\boldsymbol{x}_G}$ and set it to zero in order to find the $\boldsymbol{x}_G$ that points most-down. Then, $\boldsymbol{y}_G = \boldsymbol{z}_G \times \boldsymbol{x}_G$ and we can use these three axes to construct our desired rotation matrix $\hat{\mathbf{R}}$.

With $\hat{\mathbf{R}}$, we use the same cost function from [41] to constrain the robot to face this direction.

$$c(\mathbf{R}_i^R) = \log(\hat{\mathbf{R}}^{-1}\mathbf{R}_i^R)^\vee$$

where $\log(\cdot)$ is the logarithmic map and $\vee$ is the operator that takes a skew-symmetric matrix to a vector.



(a) (b)

Figure 3: (a) A reach-to-point task around an obstacle. We recorded 29 trials of a reach-to-point task, with varying target points, camera poses, and starting positions. (b) The predicted and measured human's trajectory for one trial. Here, the person chose to take a wider path around the obstacle than necessary.

As a simplifying assumption to improve planning efficiency, we only compute $\hat{\mathbf{R}}$ once at each planning cycle using current observation of both the robot and agent. Since we run many iterations closed-loop, the robot will continue to face towards the human's position.

2.4.2 *Modeling the human*

We use a reduced, but similar, set of cost functions for the external agent, *i.e.*, the human, $c^A(\boldsymbol{\xi}^A) = \sum_{i=0}^{T} c^A(\mathbf{q}_i^A, \dot{\mathbf{q}}_i^A, \ddot{\mathbf{q}}_i^A)$. We model the human hand as a floating sphere and the parameters λ_A are determined through a set of 29 recorded reach-to-point tasks. In the task of handover where the robot and human are similar heights, we propose that a floating sphere is a sufficient model for the human. For other collaborative tasks, such as handover at significantly different heights, the human's morphology and the kinematic feasibility of the task would be important.

Our model for optimal human reaching is a parametrized quadratic cost function that is nearly symmetrical to the robot. We model the human as a sphere representing their hand location, so we omit the constraint of joint limits. We also assume that a cooperative human will rotate their hand to meet the robot comfortably, and so we omitted the rotational constraint as well. To evaluate our model, we recorded a set of 29 reach-to-point tasks around an obstacle using the Microsoft

Azure Kinect DK and its included body tracking SDK. Between trials, we randomly changed the target position, the human's starting pose, and the camera height. See Figure 3 for an example of our setup.

For each trial, we calculated our prediction error with

$$\text{Loss} = \sum_{t=0}^{T} \frac{1}{T-(t+1)} \sum_{i=t+1}^{T} \left\| \mathbf{t}_{\text{predicted}} - \mathbf{t}_{\text{measured}} \right\|^2 \tag{13}$$

where T is the total number of time steps it takes the human to reach the target and x is the position of the hand. This loss represents the average distance between the corresponding real and predicted hand poses, which we then average over all MPC steps. We used 26 of our trials to tune our human model and performed a grid search over $6,561$ parameter configurations to minimize error. We then evaluated the parameters on the remaining 3 datasets. Our average loss on the training set 7.54cm and our average loss on the evaluation set is 9.63cm and a standard deviation of 2.41cm.

2.4.3 *Modeling the robot-agent collaboration*

At the end of the trajectory, the robot and the uncontrolled agent should meet. To enforce this, we encourage their end effector positions to be as close to each other as possible,

$$c^I(\boldsymbol{\xi}^R, \boldsymbol{\xi}^A) = \left\| \mathbf{t}_T^R - \mathbf{t}_T^A \right\|^2 \tag{14}$$

where $\mathbf{t}_T^R$ denotes the position of the robot's end-effector at the final time step and $\mathbf{t}_T^A$ denotes the position of the human's hand at that final time step (see the notation around forward kinematics in Section 2.4.1).

The interaction term defines interaction only at the end of the trajectory (the behavior is finished once the interaction occurs). In general, it is unclear when (time-wise) this interaction should occur, so choosing a single T is challenging, even more-so when re-optimizing the system and rejecting system perturbations within an MPC loop. The next section designs a sparse reward motivated by reinforcement

learning settings to eliminate this problem, enabling spatially consistent behavior using a time-parameterized trajectory model.

2.4.4 *Time Independence through Sparse Rewards*

When two agents collaborate without explicit time synchronization, their interaction and behavior is often a function of combined state and not tied to a specific clock. For example, when handing over an object, both participants time their behaviors based on the observed state of the other, continually readjusting and aiming primarily to just meet in the middle. The behavior is state-dependent and not explicitly timed.

We account for deviations from the planner's output by continually re-optimizing with a fixed-time horizon at each successive time step. However, as the two agents approach each other, this fixed horizon becomes restrictive. Suppose the horizon is three seconds in the future. Placing the interaction term perpetually at the fixed time horizon means that the model will always want to interact exactly three seconds in the future, independent of where it finds itself, leading to an exponential slowdown in its behavior.

We counteract the slowdown by adding an additional distance-based reward term weighted by λ_{reward} to the robot-agent collaboration cost defined in Eq. 14 at every point on the trajectory.

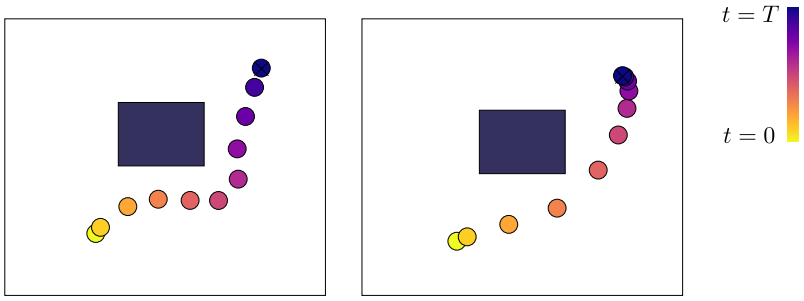
$$c(\boldsymbol{\xi}^R, \boldsymbol{\xi}^A) = c(\mathbf{t}_T^R, \mathbf{t}_T^A) + \lambda_{\text{reward}} \sum_{i=0}^{T} r(\mathbf{t}_i^R, \mathbf{t}_i^A), \tag{15}$$

where the reward $r(\mathbf{t}_i^R, \mathbf{t}_i^A)$ at step i is defined as

$$r(\mathbf{t}_i^R, \mathbf{t}_i^A) = 1 - e^{\frac{-\left\|\mathbf{t}_i^R - \mathbf{t}_i^A\right\|^2}{2\sigma^2}}. \tag{16}$$

This reward term can also apply to a single agent moving toward a fixed target, where $\left\|\mathbf{t}_i^{\text{Agent}} - p^{\text{Target}}\right\|$ would replace $\left\|\mathbf{t}_i^R - \mathbf{t}_i^A\right\|$.

The reward is motivated by the types of sparse rewards used in reinforcement learning [136]. We are rewarding the agents for converging, and, since our goal is to minimize cost, we phrase reward as



(a) Without reward term (b) With reward term

Figure 4: Comparison motion generation for a reach-to-point task around an obstacle both with and without the proposed sparse reward term. The dots represent subsequent positions. With each MPC step, the agent starts closer to its goal. Our reward terms encourages the agent to speed up as a function of their relative distance to the goal arrive at the goal in less time than the planning horizon.

negative cost. Such a reward can be modeled as an upside down radial basis function over the distance between the robot's end-effector and the interacting agent, *i.e.*, one when the two are far apart and decreasing to zero as they draw closer.

This rewards the robot for getting within touching distance of the interacting agent (and visa-versa), but does not penalize the pair for having to be far from each other earlier in the trajectory due to competing smoothness criteria. Since we have a fixed finite-horizon, without loss of generalization, we can shift each of these reward terms up by a constant so its minimum value is zero as given in the equation. The effect can be seen in Fig. 4, which shows how this results in a more temporally-consistent trajectory. Specifically, in the absence of perturbations, the trajectory traced out by MPC's replan-execute loop is more consistent with the original trajectory initially planned at the first time step.

For safety reasons, the uncontrolled agent may stop before reaching the robot. The reward term as formulated in Eq. 16 would cause the robot to slow-down exponentially because the human policy predicts that the human will keep moving. To prevent this, we would ideally have a phase estimator that can determine when the human has stopped and switch to rewarding the robot for reaching the human's current position. As an approximation in our implementation,

the human and robot are both rewarded for reaching each others' starting points, as determined at the beginning of each MPC step.

With this shift upward, our formulation of the reward (as cost) becomes identical to the Welsch robust estimator [62]. Although the reward is not a nonlinear least squares term, it can be minimized using a form of iteratively re-weighted least-squares [180] using weights given by the Radial Basis Function (RBF) $w(r) = e^{\frac{-r^2}{2\sigma^2}}$ where $r = \|\mathbf{t}_i^R - \mathbf{t}_i^A\|$ in this case. An implementation would replace these Welsch robust estimator objective terms with weighted least squares terms of the form $wr^2 = w\|\mathbf{t}_i^R - \mathbf{t}_i^A\|^2$, and re-evaluating the weight w after each subproblem has converged.

These reward terms reward the system for reaching the interaction point early. As above, associated with reaching the interaction point should be a velocity penalty bringing the system to a stop. Whereas before, it sufficed to add just a single velocity penalty to the terminal potential (stop at the end) we now must also add intermediate velocity penalties preparing for the possibility of stopping early.

2.5 IMPLEMENTATION DETAILS

Similar to [105], we used GTSAM as a fast optimizer to minimize the cost at each MPC-step. We used the real-time body tracking SDK on the Microsoft Azure Kinect DK to obtain human pose estimates, and we are able to run our algorithm to perform a human handover in real time on a Franka Emika Panda arm.

On a workstation with an *3.4ghz* Intel© Core™ *i7* and *32GB* of RAM running Ubuntu 18.04, we obtained poses at a rate of 30hz. We use DART [144] to calibrate the robot configuration into the Azure frame, so we can obtain both human and robot starting positions at each MPC step.

We run both our optimizer and DART on the same workstation running Ubuntu 16.04 and equipped with an *3.7ghz* Intel© Core™ and *32GB* of RAM. We obtain DART's positional estimates at 10hz, and we are able to run our optimizer with Levenberg-Marquardt between 7hz and 8hz.

When the Franka is within a minimum threshold–we used 10cm–it engages the gripper and tries to grasp the object. If it misses and closes all the way, the gripper re-opens and the planner resumes trying to engage in the handover until it succeeds. We found the robot to miss the handover when the human moves too quickly for the body tracker to maintain a stable estimate.

2.6 EXPERIMENTS

We ran a set of experiments exploring: (1) How do our method's generated trajectories compare to those produced by baseline methods? (2) How robust is our algorithm to noisy sensors? and (3) Can our proposed method be used in a real world setting?

Table 1: Algorithmic benchmarks (↑ denotes higher is better and ↓ denotes lower is better): our algorithm is best able to approximate the timing of the uncontrolled agent. The attractor-based algorithm produces trajectories with significantly greater acceleration and jerk than both the robot-only and our algorithm. The robot-only algorithm outperforms ours by a small margin in reducing acceleration and jerk, but at the cost of producing much longer trajectories.

Metric		Robot only	Attractor	Ours
Handover Time (Normalized)	↓	1.33 ± 0.27	1.30 ± 0.26	$\mathbf{1.20 \pm 0.26}$
Trajectory Length Error	↓	0.35 ± 0.27	0.37 ± 0.29	$\mathbf{0.27 \pm 0.22}$
Acceleration (cm/s^2)	↓	$\mathbf{4.33 \pm 1.96}$	7.83 ± 3.96	4.72 ± 1.37
Jerk μ (cm/s^3)	↓	$\mathbf{6.25 \pm 2.99}$	10.06 ± 5.55	6.36 ± 1.88

Our algorithm predicts the motion and dynamics of the uncontrolled agent and reacts accordingly. In order to evaluate each component, we benchmarked our algorithm against two different baselines:

Robot only: A planner that only accounts for the Euclidean position of the uncontrolled agent. At each time step, the robot optimizes a trajectory around any obstacles to match its end effector position with the other agent's end effector position.

Attractor: A planner that applies an attractor-based policy to both end effectors. This policy assumes the two arms will move toward each other at each time step. When obstacles are present, they act as repellent forces, opposing the attracting force. We implemented this method by using our same algorithm with a very short time horizon,

i.e., $T = 5$, which is the shortest trajectory supported by our low-level controller.

See our video submission for an example of the simulated environment. The robot and uncontrolled agent start on opposite sides of a non-convex obstacle, the position and shape of which we randomized for each trial. To evaluate the algorithms without bias, we independently planned the uncontrolled trajectory to go from a randomized location on the opposing side of the obstacle to a randomized point in the robot's reachable space, while also avoiding the obstacles. We accomplish this by minimizing our same velocity, obstacle-avoidance, and acceleration costs for the hand, while also adding a cost term with high λ to constrain the hand to our randomly chosen start and end positions, as in [105]. We augmented the plans with noise drawn from a uniform distribution to de-bias the uncontrolled motion from the planner.

We then replayed the uncontrolled trajectory for each robot policy. At the end of the uncontrolled trajectory, the agent pauses and waits for the robot. We modeled the agent as a ball with a 10cm radius and as such, we considered a trial to be successful if the robot was able to plan a trajectory where its end effector was within 10cm of the agent in under twice the uncontrolled trajectory length.

We ran 300 trials. Qualitatively, we observed that the randomized obstacle and randomized uncontrolled trajectory led to many ill-formed handovers, such as ones where the uncontrolled trajectory goes through the obstacles. The robot-only algorithm best handled these ill-formed trajectories because it is able to ignore whether the human is in an incorrect configuration. It succeeded in 62% of the trials. Our algorithm succeeded in 57% of the trials and the attractor policy succeeded in 43% of the trials. It is important to note that our framework assumes that uncontrolled agents are co-operative and reactive. To fairly compare the three policies, we used identical trajectories for the uncontrolled agent, but this precluded the possibility of the uncontrolled agent's policy being reactive to the robot and therefore violates our cooperative assumption. With a cooperative partner, we expect our policy to outperform the robot-only policy in success rate.

Table 2: Robustness metrics: we evaluate our robustness to measurement noise by determining, for a given amount of measurement noise, the percentage of handovers that can be completed within twice the time of the uncontrolled trajectory

Noise σ (cm)	2	5	7	10	15
% Successful	100	100	98	86	66

We evaluated the algorithms on the set of trials on which they mutually succeeded and adopted four different metrics, *i.e.*, *Success rate, trajectory length error, acceleration*, and *jerk*, for evaluation. We define the *trajectory length error* as $|1 - T_{success}/T_{uncontrolled}|$ where $T_{success}$ is the time it takes to finish a successful action and $T_{uncontrolled}$ is the length of the uncontrolled trajectory. Quantitative results are in Table 1.

Both qualitatively and quantitatively, we saw that the *attractor* algorithm leads the robot to jerk heavily when the agent's path around the obstacle is non-obvious. Meanwhile, the *robot-only* planner tends to wait to move until the path around the obstacle is unobstructed, leading it to take longer to reach the agent. Our algorithm is able to smoothly predict the agents path. We also saw our algorithm produces lower *trajectory length error* than the others–meaning our algorithm is better able to match the length of the uncontrolled trajectory. See our video for a demonstration.

We also evaluated our algorithm's robustness to measurement noise. In our real-robot experiments, we observed that our planner failed when the calibration and/or body tracker were misaligned. To measure this, we planned a randomized uncontrolled trajectory in the same fashion as before. However, we also introduced Gaussian noise with increasing σ into the robot's perception of the trajectory. We ran 50 trials and measured how often the robot could intersect the agent at its *actual* location within twice the uncontrolled trajectory length. Results with varying σ are in Table 2.

As shown in Figure 1, we are able to run our algorithm on a real robot using the setup described in Section 2.5. See our video for more examples.

2.7 CONCLUSION AND FUTURE WORK

We proposed an MPC approach for multi-agent collaboration problems that simultaneously optimizes motion plans for a robot and an (uncontrolled) human in order to enable coordination on cooperative tasks, with an application to human-robot handovers in obstacle-rich environments. We presented a novel theoretical framework and demonstrated its effectiveness through both simulated and real-robot experiments. This framework assumes access to a model for the human collaborator, and future work might learn such a model from data, for example, via Inverse Optimal Control [185], which has been successfully applied to motion prediction in the past [79]. In addition, our approach cannot generate longer-term plans across manipulations, as in [165]; in the future, we will develop approaches which maintain reactivity but allow for switching between discrete modes, such as when the human is waiting for the robot at the end of a handover, possibly via the Robust Logical-Dynamical Systems formalism [119]. We also intend to apply our formal system to other coordination problems explored in the literature, such as motion in a crowd [3] and camera control [17, 128].

Part II

END TO END LEARNING

In the following chapter, I discuss our work that uses large-scale imitation learning to address many of the common challenges in analytic motion generation techniques. This research was a significant step forward in employing large-scale learning for robot motion, introducing a dataset with millions of demonstations–300 times larger than prior work. The core innovation of this work lies in its ability to encode complex long-horizon motion planning intuition directly into a learned policy, thereby enabling a reactive local that behaves much like a long-horizon planner.

Motion Policy Networks (MπNets) are particularly effective in scenarios where rapid decision-making is crucial. Our original goal was to mimic the trajectory optimizer used in [48] while avoiding the need for expesive scene reconstruction. While the research eventually focused more on collision-free reaching, the initial problem set-up led us to focus on reaction speed in fast changing environments. While many local policies, *e.g.*[14, 173], are effective at fast changing goals, these methods do not perform as well in fast changing environments with unpredictable elements.

This chapter delves into how we were able to learn an effective policy that can safely move through novel environments in real time without any explicit scene reconstruction. Moreover, I will discuss the benefits, challenges, and limitations of traditional analytic methods, and compare the performance of MπNets against other state-of-the-art robotic motion planning techniques. While MπNets shows impressive performance in these baselines, there are many situations in which it fails due to poor data coverage or other factors. In this chapter, I will discuss the benefits and challenges of using MπNets for end-to-end motion generation.

3

MOTION POLICY NETWORKS

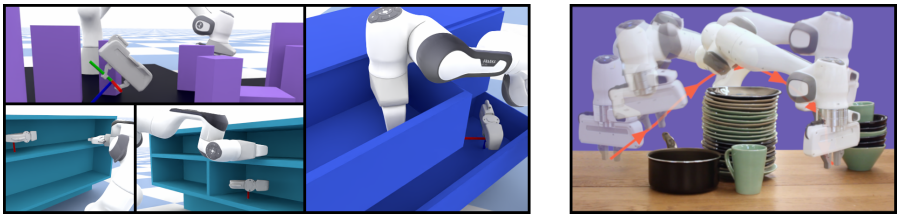


Figure 5: MπNets are trained on a large dataset of synthetic demonstrations (*left*) and can solve complex motion planning problems using raw point cloud observations (*right*).

3.1 INTRODUCTION

Generating fast and legible motions for a robotic manipulator in unknown environments is still an open problem. Decades of research have established many well-studied algorithms, but there are two practical issues that prevent motion planning methods from being widely adopted in industrial applications and home environments that require real-time control. First, it is challenging for any single approach to satisfy multiple planning considerations: speed, completeness, optimality, ease-of-use, legibility (from the perspective of a human operator), determinism, and smoothness. Second, existing approaches enforce strong assumptions about the visual obstacle representations—such as accurate collision checking in configuration space [87] or the availability of a gradient [132, 145, 173]—and hence require expensive intermediate processing of sensor readings to operate in novel scenes.

Some planners, such as RRT [86], are useful to quickly find a feasible path. Other methods are useful to find an optimal path, either deterministically on discrete graphs [58, 92] or asymptotically in continuous spaces [72, 154]. Some planning techniques use Neural Networks [65, 125] to improve sampling efficiency, thus speeding up the

planning process. Optimization-based methods [132, 145] are useful to encode other objectives, such as smoothness, at the expense of guarantees. Going further, recent motion generation frameworks [13, 173] eschew long-term reasoning in exchange for quick local decisions under the assumption that they will lead to globally acceptable paths.

Each of these methods requires a known environment model and perfect state estimation. In practice, one would have to create a scene representation, which could be a static or dynamic mesh, an occupancy grid [65, 75], a signed distance field, etc. Reconstruction systems such as SLAM and KinectFusion [111] have a large system start-up time, require a moving camera to aggregate many viewpoints, and ultimately require costly updates in the presence of dynamic objects. Recent implicit deep learning methods like DeepSDF [118] and NeRF [99] are slow or do not yet generalize to novel scenes. Methods such as SceneCollisionNet [36] provide fast collision checks but have not yet been shown to generalize to challenging environments beyond a tabletop. Other RL-based methods learn a latent representation from observations but have only been applied to simple 2D [70, 159] or 3D [155] environments in simulation.

We present *Motion Policy Networks (MπNets)*, a novel method for learning an end-to-end policy for motion planning. Our approach circumvents the challenges of traditional motion planning and is flexible enough to be applied in unknown environments. Our contributions are as follows:

- We present a large-scale effort in neural motion planning for manipulation. Specifically, we learn from over 3 million motion planning problems across over 500,000 instances of three types of environments, nearly 300x larger than prior work [125].
- We train a reactive, end-to-end neural policy that operates on point clouds of the environment and moves to task space targets while avoiding obstacles. Our policy is significantly faster than other baseline configuration space planners and succeeds more than local task space controllers.

- On our challenging dataset benchmarks, we show that MπNets is nearly 46% more successful at finding collision-free paths than prior work [125] without even needing the full scene collision model.
- Finally, we demonstrate *sim2real* transfer to real robot partial point cloud observations.

3.2 RELATED WORK

Global Planning: Robotic motion planning typically splits into three camps: search, sampling, and optimization-based planning. Each algorithmic family has benefits and drawbacks. Search-based planning algorithms, such as A* [58, 91, 92] are complete, fast, and optimal in discrete domains. Sampling-based methods, e.g. [72, 86], operate in continuous domains, but are only probabilistically complete, *i.e.*find a solution with probability 1. Some such methods are also asymptotically optimal [51, 72, 154], but within practical time limitations produce sub-optimal—and sometimes erratic—paths. Motion Optimization [131, 132, 145] can produce smooth paths to a goal but is prone to local minima. Without careful system design, often on a per-task basis, Motion Optimization can fail to find the optimal solution or sometimes any solution at all.

Search-based planning algorithms, such as A* [58, 91, 92], discretize the state space and perform a graph search to find an optimal path. While the graph search can be fast, complete, and guaranteed optimal, the requirement to construct a discrete graph hinders these algorithms in continuous spaces and for novel problems not well covered by the current graph. Sampling-based planners [86] function in a continuous state space by drawing samples and building a tree. When the tree has sufficient coverage of the planning problem, the algorithm traverses the tree to produce the final plan. Sampling-based planners are continuous, probabilistically complete, *i.e.*find a solution with probability 1, and some are even *asymptotically optimal* [51, 72,

154], but under practical time limitations, their random nature can produce erratic—though valid—paths.

Both of the aforementioned planner types are designed to optimize for path length in the given state space (*e.g.*configuration space) while avoiding collisions. An optimal path in configuration space is not necessarily optimal for the end effector in cartesian space. Human motion tends to minimize hand distance traveled [168], so what appears optimal for the algorithm may be unintuitive for a human partner or operator. In the manipulation domain, goals are typically represented in end effector task space [126, 158]. In a closed loop setting with a moving target, the traditional process of using IK to map task to configuration space can produce highly variable configurations, especially around obstacles. Motion Optimization [131, 132, 145] on the other hand, generates paths with non-linear optimization and can consider multiple objectives such as smoothness of the motion, obstacle avoidance, and convergence to an end effector pose. These algorithms require careful tuning of the respective cost functions to ensure convergence to a desirable path and are prone to local minima. Furthermore, non-linear optimization is computationally complex and can be slow for difficult planning problems.

Local Control: In contrast to global planners, local controllers have long been applied to create collision-free motions [13, 76, 129, 173]. While they prioritize speed and smoothness, they are highly local and may fail to find a valid path in complex environments. We demonstrate in our experiments that MπNets are more effective at producing convergent motions in these types of environments, including in dynamic and in partially observed settings.

Imitation Learning: Imitation Learning [116] can train a policy from expert demonstrations with limited knowledge of the expert's internal model. For motion planning problems, we can apply imitation learning and leverage a traditional planner as the expert demonstrator—with a perfect model of the scene during training—and learn a policy that forgoes the need for an explicit scene model at test time. Popular imitation learning methods include Inverse Reinforcement Learning [112, 142, 184] and Behavior Cloning [4, 121]. Our method seeks to

overcome the common challenges of Behavior Cloning by specifically designing a learnable expert, increasing the scale and variation of the data, and using a sufficiently expressive policy model.

Inverse Reinforcement Learning [112, 142, 184] typically assumes expert optimality and learns a cost function accordingly, whereas Behavior Cloning [4, 121] directly learns the state-action mapping from demonstrations, regardless of the expert's optimality. We thus employ behavior cloning because producing optimal plans for continuous manipulation problems is challenging. Recent work demonstrates behavior cloning's efficacy for fine-grained manipulation tasks, such as chopstick use [74] and pick-and-place [97]. For long-horizon tasks like ours, however, distributional shift and data variance can hinder behavior cloning performance. Distribution shift during execution can lead to states unseen in training data [74]. Complex tasks often have a long tail of possible action states that are underrepresented in the data, leading to high data variance [32]. There are many techniques to address these challenges through randomization, noise injection, regret optimization, and expert correction [74, 89, 137, 139, 140]. These techniques, however, have not been demonstrated on a problem of our scale and complexity (see Section 3.4.2 for details on the range of data).

Neural Motion Planning: Many deep planning methods [22, 65, 82, 179] seek to learn efficient samplers to speed up traditional planners. Motion Planning Networks (MPNets) [125] learn to directly plan through imitation of a standard sampling-based RRT* planner [72] and is used in conjunction with a traditional planner for stronger guarantees. While these works greatly improve the speed of the planning search, they have the same requirements as a standard planning system: targets in configuration space and an explicit collision checker to connect the path. Our work operates based on task-space targets and perceptual observations from a depth sensor without explicit state estimation.

Novel architectures have been proposed, such as differentiable planning modules in Value Iteration Networks [159], transformers by Chaplot, Pathak, and Malik [23] and goal-conditioned RL policies [45].

These methods are challenging to generalize to unknown environments or have only been shown in simple 2D [70] or 3D settings [155]. In contrast, we illustrate our approach in the challenging domain of controlling a 7 degrees of freedom (DOF) manipulator in unknown, dynamic environments.

3.3 LEARNING FROM MOTION PLANNING

3.3.1 *Problem Formulation*

MπNets expect two inputs, a robot configuration q_t and a segmented, calibrated point cloud z_t. Before passing q_t through the network, we normalize each element to be within $[-1, 1]$ according to the limits for the corresponding joint. We call this $q_t^{\|\cdot\|}$. The point cloud is always assumed to be calibrated in the robot's base frame, and it encodes three segmentation classes: the robot's current geometry, the scene geometry, and the target pose. Targets are inserted into the point cloud via points sampled from the mesh of a floating end effector placed at the target pose.

The network produces a displacement within normalized configuration space $\dot{q}_t^{\|\cdot\|}$. To get the next predicted state $\hat{q}_{t+1}$, we take $q_t^{\|\cdot\|} + \dot{q}_t^{\|\cdot\|}$, clamp between $[-1, 1]$, and unnormalize. During training, we use $\hat{q}_{t+1}$ to compute the loss, and when executing, we use $\hat{q}_{t+1}$ as the next position target for the robot's low-level controller.

3.3.2 *Model Architecture*

The network consists of two separate encoders, one for the point cloud and one for the robot's current configuration, as well as a decoder, totaling 19M parameters. Our neural policy architecture is visualized in Fig. 6. We use PointNet++ [123] for our point cloud encoder. PointNet++ learns a hierarchical point cloud representation and can encode a point cloud's 3D geometry, even with high variation in sampling density. PointNet++ architectures have been shown

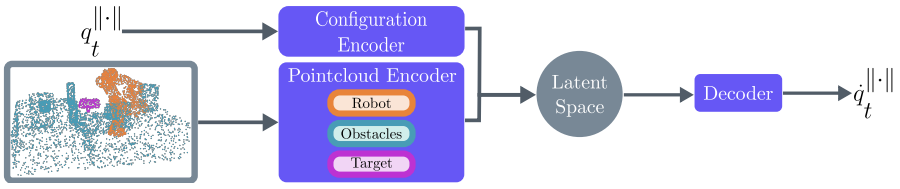


Figure 6: MπNets encodes state as a normalized robot configuration and segmented point cloud with three classes for the robot, the obstacles, and the target. The policy outputs a displacement in normalized joint space, which can then be applied to the input before unnormalizing to get q_{t+1}.

to be effective for a variety of point cloud processing tasks, such as segmentation [123], collision checking [36], and robotic grasping [103, 108]. Additionally, PointNet++ includes PointNet as a subcomponent. PointNet is effective at processing partially observed point clouds, even when trained exclusively with fully-observed scenes [122]. The robot configuration encoder and the displacement decoder are both fully connected multilayer perceptrons.

Our PointNet++ architecture has three set abstraction groups followed by three fully connected layers. The first set abstraction layer performs iterative furthest point sampling to construct a set of 512 points, then it does a grouping query within 5cm of at most 128 points. Finally, there is a local PointNet [122] made up of layers of size 4, 64, 64, 64 respectively. The second set abstraction is lower resolution, sampling 128 furthest points and then grouping at most 128 points within a 30cm radius. The corresponding PointNet is made up of layers of size 64, 128, 128, and 256 respectively. Our third set abstraction layer skips the furthest point sampling, groups all points together, and uses a final PointNet with layers of size 256, 512, 512, 1024 respectively. Finally, after the set abstraction layers, we have three fully connected layers with 4096, 4096, and 2048 dimensions respectively. In between these layers, we use group norm and Leaky ReLU.

The output of our point cloud encoder is a 2048 dimensional embedding. The robot configuration encoder and the displacement decoder are both fully connected multilayer perceptrons with Leaky ReLU activation functions [94]. The robot configuration encoder maps our 7-dimensional input to a 64-dimensional output and has four hid-

den layers with 32, 64, 128, and 128 dimensions respectively. The displacement decoder maps the combined embeddings from the point cloud and robot configuration encoders, which together have 2112 dimensions, to the 7 dimensional normalized displacement space. The decoder has three hidden layers with 512, 256, and 128 dimensions respectively. Our entire architecture together has 19 million parameters.

3.3.3 *Loss Function*

The network is trained with a compound loss function with two constituent parts: a behavior cloning loss to enforce accurate predictions and a collision loss to safeguard against catastrophic behavior.

GEOMETRIC LOSS FOR BEHAVIOR CLONING To encourage alignment between the prediction and the expert, we compute a geometric loss across a set of 1024 fixed points along the surface of the robot.

$$L_{BC}(\hat{\Delta}q_t) = \sum_i \|\hat{x}^i_{t+1} - x^i_{t+1}\|_2 + \|\hat{x}^i_{t+1} - x^i_{t+1}\|_1$$
$$\text{where} \quad \hat{x}^i_{t+1} = \phi^i(q_t + \hat{\Delta}q_t), \quad x^i_{t+1} = \phi^i(q_{t+1}) \tag{17}$$

$\phi^i(\cdot)$ represents a forward kinematics mapping from the joint angles of the robot to point i defined on the robot's surface. The loss is computed as the sum of the L1 and L2 distances between corresponding points on the expert and the prediction after applying the predicted displacement. By using both L1 and L2, we are able to penalize both large and small deviations.

We use a geometric, task-space loss because our goal is to ensure task-space consistency of our policy. Configuration space loss appears in prior work [125], but does not capture the accumulated error of the kinematic chain as effectively (see Appendix K).

COLLISION LOSS In order to avoid collisions–a catastrophic failure–we apply an additional hinge-loss inspired by motion optimization [48].

$$
\begin{aligned}
L_{\text{collision}} &= \sum_i \sum_j \|h_j(\hat{x}^i_{t+1})\|_2 \\
\text{where } h_j(\hat{x}^i_{t+1}) &= \begin{cases} -D_j(\hat{x}^i_{t+1}), & \text{if } D_j(\hat{x}^i_{t+1}) \leqslant 0 \\ 0, & \text{if } D_j(\hat{x}^i_{t+1}) > 0 \end{cases}
\end{aligned} \tag{18}
$$

The synthetic environments are fully-observable during training, giving us access to the signed-distance functions (SDF), $\{D_j(\cdot)\}_j$, of the obstacles in each scene. For a given closed surface, its SDF maps a point in Euclidean space to the minimum distance from the point to the surface. If the point is inside the surface, the function returns a negative value.

3.4 PROCEDURAL DATA GENERATION

3.4.1 *Large-scale Motion Planning Problems*

Each planning problem is defined by three components: the scene geometry, the start configuration, and the goal pose. Our dataset consists of randomly generated problems across all three components, totaling 3.27 million problems in over 575,000 environments. We have three classes of problems of increasing difficulty: a cluttered tabletop with randomly placed objects, cubbies, and dressers. Representative examples of these environments are shown in Fig. 5. Once we build these environments, we generate a set of potential end effector targets and corresponding inverse kinematics solutions. We then randomly choose pairs of these configurations and verify if a plan exists between them using our expert pipeline, as detailed further in Sec. 3.4.2 and in Appendix E.

3.4.2 *Expert Pipeline*

Our expert pipeline is designed to produce high-quality demonstrations we want to mimic, *i.e.*trajectories with smooth, consistent motion and short path lengths. Here, *consistency* is meant to describe quality and repeatability of an expert planner—see Appendix C for further discussion. We considered two candidates for the expert - the *Global Planner* which is a typical state-of-the-art configuration space planning pipeline [153] and a *Hybrid Planner* that we engineered specifically to generate consistent motion in task space. For both planners, we reject any trajectories that produce collisions, exceed the joint limits, exhibit erratic behavior (*i.e.*high jerk), or that have divergent motion (*i.e.*final task space pose is more than 5cm from the target).

Global Planner consists of off-the-shelf components of a standard motion planning pipeline–inverse kinematics (IK) [40], configuration-space AIT* [153], and spline-based, collision-aware trajectory smoothing [60]. For a solvable problem, as the planning time approaches infinity, IK will find a valid solution and AIT* will produce an optimal collision-free path, both with probability 1. Likewise, with continuous collision checking, the smoother will produce a smooth, collision-free path. In practice, our dataset size goal—we generated 6.54M trajectories across over 773K environments—dictated our computation budget and we tuned the algorithms according to this limit. We attempted IK at most 1000 times, utilized an AIT* time out of 20s, and employed discrete collision checking when smoothing. Most commonly, the pipeline failed when AIT* timed out or when, close to obstacles, the smoother's discrete checker missed a collision, thereby creating invalid trajectories. The Global Planner is composed of widely used off-the-shelf components. We first use inverse kinematics to convert our task space goals to configuration space, followed by AIT* [153] in configuration space, and finally, spline-based, collision-aware trajectory smoothing [60]. We use IKFast [40] for inverse kinematics, OMPL [156] for AIT*, and Pybullet Planning for the smoothing implementation [53]. To manage the compute load when generating a

large dataset of trajectories, we employed a time-out with AIT* of 20 seconds.

Hybrid Planner is designed to produce consistent motion in task space. The planner consists of task-space AIT* [153] and Geometric Fabrics [173]. AIT* produces an efficient end effector path and Geometric Fabrics produce geometrically consistent motion. The end effector path acts as a dense sequence of waypoints for a sequence of Geometric Fabrics, but as the robot moves through the waypoints, the speed can vary. To promote smooth configuration space velocity over the final trajectory, we fit a spline to the path and retime it to have steady velocity. As we discuss in Sec. 3.5.1, Geometric Fabrics often fail to converge to a target, so we redefine the planning problem to have the same target as the final position of the trajectory produced by the expert. Inspired by [2], we call this technique *Hindsight Goal Revision (HGR)* and demonstrate its importance in Sec. 3.5.4. Using the *Hybrid Planner*, we generated 3.27 million trajectories across 576,532 environments. The Hybrid Expert is designed to produce consistent motion in task space. We start by using AIT* [153] with a 2 second timeout to plan for a floating end effector, *i.e.*one not attached to a robot arm, and then use Geometric Fabrics [173] to follow the path. Geometric Fabrics are deterministic and geometrically consistent [173] local controllers, but they struggle to solve the problems in our dataset without assistance from a global planner. Geometric Fabrics are highly local, and even with dense waypoints given by a global planner, they can run into local minima, which in turn generate trajectories with highly variable velocity. We use a combination of spline-based smoothing and downsampling [59] to create a consistent configuration space velocity profile across our dataset.

Consistency We use the term *consistency* to describe a qualitative characteristic of a planner and its learnability. Specifically, we use it to describe two quantities: 1) expert quality and 2) repeatability of the planner. Mandlekar et al. [97] demonstrate how Imitation Learning performance varies depending on expert quality. Among the metrics they use to describe expert quality, they demonstrate the importance of expert trajectory length. MπNets employs task-space goals, and

the *Hybrid Planner* produces shorter task-space paths. Across our test dataset of global and hybrid solvable problems, the *Hybrid Planner*'s end effector paths average 57cm ± 31cm and the total orientation distance traveled is 95° ± 52°. Meanwhile, the *Global Planner*'s paths average 61cm ± 39cm and 113° ± 55°, respectively.

Repeatable input-output datasets are important for deep learning systems. Prior works have shown that deep learning systems deteriorate or require more data when using noisy labels [69, 101]. Both the *Global Planner* and *Hybrid Planner* are sampling-based planners and do not produce repeatable paths by their very nature. Yet, the *Hybrid Planner* uses sampling to plan in a lower-dimensional state space—6D pose space—while the *Global Planner* samples in 7D configuration space. We use a naive sampler, so the lower dimensionality of the *Hybrid Planner's* sampler implies that its typical convergence rate will be faster. After planning, the *Hybrid Planner* employs Geometric Fabrics [173] to follow the task-space trajectory. Geometric Fabrics are deterministic, which further promotes repeatability in the final, configuration space trajectories. Meanwhile, the *Global Planner* uses a randomized smoothing algorithm that is not deterministic. Taking these individual components together, we expect the *Hybrid Planner's* solutions on similar problems to be typically more alike than the *Global Planner's* solutions to the same problems.

3.5 EXPERIMENTAL EVALUATION

We evaluate our method with problems generated from the same distribution as the training set. Within the test set, each problem has a unique, randomly generated environment, as well as a unique target and starting configuration. None of the test environments, starting configurations, nor goals were seen by the network during training. Our evaluations were performed on three test sets: a set of problems solvable by the *Global Planner*, problems solvable by the *Hybrid Planner*, and problems solvable by both. Each test set has 1800 problems, with 600 in each of the three types of environments.

Quantitative Metrics: To understand the performance of a policy, we roll it out until it matches one of two termination conditions: 1) the Euclidean distance to the target is within 1cm or 2) the trajectory has been executed for 20 s (based on consultations with the authors of [173] and [13]). We consider the following metrics:

- *Success Rate* - A trajectory is considered a success if its final position and orientation target errors are below 1 cm and 15° respectively, and there are no physical violations.
- *Time* - We measure the wall time for each *successful* trajectory. We also measure *Cold Start (CS) Time*, the average time to react to a new planning problem.
- *Rollout Target Error* - The L2 position and orientation error (taken from [172]) between the target and final end effector pose in the trajectory.
- *Collision Rate* - The rate of fatal collisions, both self and scene collisions
- *Smoothness* - We use Spectral Arc Length (SPARC) [5] and consider a path to be smooth if its SPARC values in joint and end effector space are below -1.6.

SUCCESS RATE A trajectory is considered a success if the rollout position and orientation target errors are below 1 cm and 15° respectively and there are no physical violations. To avoid erroneously passing a trajectory that ends on the wrong side of a narrow structure, we also ensure that the end effector is within the correct final volume and likewise avoids incorrect volumes. For the cubby and dresser environments, these volumes are individual cubbies or drawers.

TIME After setting up each planning problem, we measured the wall time for each *successful* trajectory. We also measure *Cold Start (CS) Time*, the average time to react to a new planning problem. While both expert pipelines have to compute the entire path, the local controllers

only need time to compute a single action. We only consider the cold-start time here, but if the new planning problem is sufficiently similar to a previous one–such as a minor change in the environment or target–a global planning system could employ an optimizer that can replan quickly [104].

ROLLOUT TARGET ERROR We calculate both position and orientation errors from the target for the final end effector pose in the trajectory. We measure position error with Euclidean distance and orientation error with the metric described by Wunsch, Winkler, and Hirzinger [172].

COLLISIONS A trajectory can have two types of fatal collisions–when the robot collides with itself or when the robot collides with the scene. When checking for collisions, we use an ensemble of collision checkers to ensure fairness. Collision checking varies across algorithmic implementations, *e.g.*our AIT* implementation uses meshes to check scene collisions, while STORM [13] and Geometric Fabrics [173] use a sphere-based approximation of the robot's geometry. A trajectory is only considered to be in collision if the entire ensemble agrees.

SMOOTHNESS We use Spectral Arc Length (SPARC) [5] to measure smoothness. Balasubramanian et al. [5] use a SPARC threshold of -1.6 as sufficiently smooth for reaching tasks. This measurement qualitatively describes the behavior of our benchmark algorithms well, so we used the same threshold for sufficiency. We therefore consider a path to be smooth if both its joint-space trajectory and end effector trajectory have SPARC values below -1.6.

3.5.1 *Comparison to Methods With Complete State*

Most methods to generate motion in the literature assume access to complete state information in order to perform collision checks. In each of the following experiments, we provide each baseline method with an oracle collision checker. When running MπNets, we use a

			Success Rate (%)			
	Soln. Time (s)	CS Time (s)	Global	Hybrid	Both	Smooth (%)
Global Planner [153]	16.46 ± 0.90	16.46 ± 0.90	100	78.44	100	51.00
Hybrid Planner	7.37 ± 2.23	7.37 ± 2.23	50.22	100	100	99.26
G. Fabrics [173]	0.15 ± 0.09	$2.4e\text{-}4 \pm 3e\text{-}5$	38.44	59.33	60.06	85.39
STORM [13]	4.03 ± 1.89	$13.4e\text{-}3 \pm 2.2e\text{-}3$	50.22	74.50	76.00	62.26
MPNets [125]						
Hybrid Expert	4.95 ± 23.51	4.95 ± 23.51	41.33	65.28	67.67	99.97
Random	0.31 ± 3.55	0.31 ± 3.55	32.89	55.33	58.17	99.96
MπNets (Ours)						
Global Expert	0.33 ± 0.08	$6.8e\text{-}3 \pm 7e\text{-}5$	75.06	80.39	82.78	89.67
Hybrid Expert	0.33 ± 0.08	$6.8e\text{-}3 \pm 7e\text{-}5$	75.78	95.33	95.06	93.81

Table 3: Algorithm performance on problems sets solvable by planner types. All prior methods use state-information and a oracle collision checker while MπNets only needs a point cloud

point cloud sampled uniformly from the surface of the entire scene. Results are shown in Table 3.

GLOBAL CONFIGURATION SPACE PLANNER The *Global Planner* is unmatched in its ability to reach a target, but this comes at the cost of average computation time (16.46s) compared to MπNets (0.33s). With a global planner, there is no option to partially solve a problem, meaning the Cold Start Time is equal to the planning time. In a real system, optimizers [104, 132, 145] could be used to quickly replan once an initial plan has been discovered. As discussed in Sec. 3.4.2, the *Global Planner* is theoretically complete but fails in practice on some of the *Hybrid Planner*-solvable problems due to system timeouts and discrete collision checking during smoothing.

HYBRID END EFFECTOR SPACE PLANNER Our *Hybrid Planner* struggles with a large proportion of problems solvable by the *Global Planner*. Yet, its solutions are both faster and smoother than the *Global Planner*. Surprisingly, MπNets trained with data from the expert *outperformed* the expert on the *Global Planner*-solvable test set. We attribute this to two features: 1) we use strict rejection sampling to reduce erratic and divergent behavior in our expert dataset and train only on the filtered data and 2) our use of Hindsight Goal Revision to turn an imperfect expert into a perfect one.

		Evaluation Set	
	Training Set	MPNets-Style	Hybrid Expert Solvable (Ours)
MPNets [125]	MPNets-Style	78.70	49.89
MπNets (Ours)	MPNets-Style	33.70	5.50
MPNets [125]	Hybrid Expert	88.90	65.28
MπNets (Ours)	Hybrid Expert	89.50	95.33

Table 4: Success rates (%) of our method compared to Motion Planning Networks (MPNets) [125] trained and evaluated on different datasets

NEURAL MOTION PLANNING Motion Planning Networks (MPNets) [125] proposed a similar method for neural motion planning, but there are a few key differences in both problem setup and system architecture. MPNets requires a ground-truth collision checker to connect sparse waypoints, plans in configuration space, and is not reactive to changing conditions. In the architecture, MPNets uses a trained neural sampler within a hierarchical bidirectional planner. The neural sampler is a fully-connected network that accepts the start, goal, and a flattened representation of the obstacle points as inputs and outputs a sample. MPNets guarantees completeness by using a traditional planner as a fallback if the neural sampler fails to produce a valid plan.

In addition to our data, we generated a set of tabletop problems, which we call *MPNets-Style*, akin to the Baxter experiments in [125], in order to fairly compare the two methods. The results of this experiment can be seen in Table 4. MπNets requires a large dataset that covers the space of test problems to achieve compelling performance, while MPNets' utilization of a traditional planning system is much more effective with a small dataset or out-of-distribution problems. However, the MPNets architecture does not scale to more complex scenes, even with more data, as we show in Fig. 7. When trained and evaluated on the Hybrid Planner-solvable dataset, MPNets succeeds in 65.28% of the test set, whereas MπNets succeeds in 95.33%, thus decreasing the failure rate by 7X. Furthermore, as we show in Table 3, using the MPNets neural sampler trained with the *Hybrid Planner* performs similarly to a uniform random sampler when both are embedded within the bidirectional MPNets planner.

				% Within			
	% Env. Coll.	% Self Coll.	% Jnt Viol.	1cm	5cm	15°	30°
G. Fabrics [173]	8.61	0.11	0.44	69.89	75.17	83.44	85.11
STORM [13]	0.93	0.11	0.25	79.81	83.54	81.57	85.41
MπNets (Ours)							
Hybrid Expert	0.94	0.00	0.00	98.94	99.72	98.22	99.00
Global Expert	13.78	0.06	0.00	98.67	99.89	97.56	99.11

Table 5: Failure Modes on problems solvable by both the global and hybrid planners

LOCAL TASK SPACE CONTROLLERS Unlike planners, which succeed or fail in a binary fashion, local policies will produce individual actions that, when rolled out, may fail for various reasons. We break down the various failure modes across the set of problems solvable by both experts in Table 5.

STORM [13] and Geometric Fabrics [173] make local decisions that can lead them to diverge from the target in complex scenarios, such as cluttered environments or those with pockets. While STORM, Geometric Fabrics, and MπNets are all local policies, STORM and Geometric Fabrics rely on human tuning to achieve strong performance. Prior environment knowledge alongside expert tuning can lead to phenomenal results, but these parameter values do not generalize. We used a single set of parameters across all test environments just as we used a single set of weights for MπNets. MπNets encodes long-term planning information across a wide variety of environments, which makes it less prone to local minima, especially in unseen environments.

On problems solvable by the *Hybrid Planner*, MπNets ties or outperforms these other methods across nearly all metrics (see Appendix Table 1). On the set of problems solvable by the *Global Planner*, MπNets target convergence rate is consistently higher, while its collision rate (11%) is worse than either STORM (1.94%) or Geometric Fabrics (7.83%) (see Appendix Table 2). Deteriorating performance on out-of-distribution problems is a typical downside of a supervised learning approach such as MπNets. However, this could be improved with a more robust expert, *e.g.*one with the consistency of our *Hybrid Planner* but

the success rate of the *Global Planner*, with finetuning, or with DAgger [139].

3.5.2 *Importance of the Expert Pipeline*

We observed that the choice of the expert pipeline affects the performance of MπNets. We trained three policies: MπNets-G with 6.54M demonstrations from the *Global Planner*, MπNets-H with 3.27M demonstrations from the *Hybrid Planner*, and MπNets-C with 3.27M demonstrations from each. MπNets-C did not exhibit improved performance over either MπNets-H or MπNets-G. When evaluated on a test set of problems solvable by the *Global Planner*, MπNets-G shows far better target convergence (97.94% vs. 87.72%) compared to MπNets-H but worse obstacle avoidance (21.94% collision rate vs. 11%). Nonetheless, MπNets-H is significantly better across all metrics when evaluated on problems solved by both experts as shown in Table 5. We hypothesize that an expert combining the properties of these two–the consistency of the *Hybrid Planner* and the generality of the *Global Planner*, would further improve MπNets's performance. We refer to MπNets-H as MπNets throughout the rest of the paper.

3.5.3 *Comparison to Methods With Partial Observations*

In addition to demonstrating MπNets' performance on a real robot system, we also compared MπNets to the *Global Planner* (AIT* [153]) in a single-view depth camera setting in simulation. We evaluated on the test set of problems solvable by both the *Global* and *Hybrid Planners*. MπNets only has a minor drop in success rate when using a partial point cloud vs. a full point cloud– from 95.06% to 93.22% though the collision rate increases from 0.94% to 3.06% due to occlusions. For this experiment, we compared to the AIT* component of our *Global Planner* alone to minimize false-positive solutions caused by the smoother's discrete collision checker (see discussion in Section 3.4.2). We used a voxel-based reconstruction akin to the standard per-

ception pipeline packaged with MoveIt [27]. In our implementation, a voxel is filled only if a 3D point is registered within it. On the same test set using the voxel representation, AIT* produces plans with collisions on 16.41% of problems. In this setting, MπNets's collision rate is over 5X smaller than that of the *Global Planner*.

3.5.4 *Ablations*

We perform several ablations to justify our design decisions. All ablations were trained using the *Hybrid Planner* dataset and evaluated on the *Hybrid Planner*-solvable test set. .

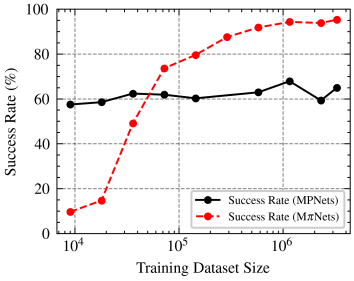


Figure 7: MπNets performance continues to increase with more training data, while MPNets performance stays relatively constant

MπNets Performance Scales with More Data As shown in Fig. 7, the performance of MπNets continues to improve with more data, although it saturates at 1.1M. Meanwhile, MPNets [125] has constant performance, demonstrating that our architecture is better able to scale with the data.

Robot Point Representation Improves Performance Instead of representing the robot by its configuration vector, we insert the robot point cloud at the specific configuration. Without this representation, the success rate decreases from 95.33% to 65.06%.

Hindsight Goal Revision Improves Convergence When trained without *HGR*, *i.e.*with the planner's original target given to the network, we see 58.11% success rate vs. 95.33% when trained with *HGR*. In particular, only 60.28% of trajectories get within 1cm of the target during evaluation.

Noise Injection Improves Robustness When we train MπNets without injecting noise into the input q_t, the policy performance decreases by 10.72%.

Training with Mean Squared Error Loss Increases Collisions When trained with a loss of mean-squared-error in configuration space, MπNets

has a similar success rate–94.56% vs. 95.33%–but the scene collision rate is significantly higher at 2.39% vs 0.89%.

Representing the Target in Point Cloud Improves Performance When trained with the target fed explicitly through a separate MLP encoder as a position and quaternion, MπNets succeeds less–88.83% vs. 95.33% when the target is specified within the point cloud. In particular, only 91.61% of trajectories get within 1cm of the target vs. 98.83% with the point cloud-based target.

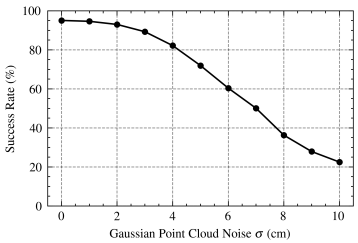


Figure 8: After injecting Gaussian noise into the point clouds, MπNets performance stays fairly constant up until $\sigma = 3$ cm when success rate is 89.28%.

Training with Collision Loss Improves Collision Rate When trained without the collision loss, MπNets collides more often–2.11% vs 0.89% when trained with the collision loss.

Training with the Configuration Encoder Improves Success Rate When trained with no robot configuration encoder, *i.e.*with only the point cloud encoder, MπNets has a success rate of 94.17% vs 95.33% when trained with both encoders.

MπNets is Robust to Point Cloud Noise Up to 3.2cm Figure 8 shows MπNets success rate on the set of problems solvable by both planners when random Gaussian noise is added to the point cloud. Model performance stays above 90% until noise reaches 3cm at which point success drops to 89.28%.

MπNets is Robust to Varying Point Cloud Shapes To evaluate performance in out-of-distribution geometries, we replaced all tabletop objects in the test set of problems solvable by the *Hybrid Planner* with randomly meshes from the YCB dataset [187]. For each tabletop primitive, we sampled a mesh from the dataset and transformed it so that the bounding boxes of the primitive and mesh were aligned and of identical size. Note that in these modified scenes, the primitives-based *Hybrid Planner* solution is still valid. MπNets succeeded in 88.33% in this YCB-tabletop test set, whereas with the original prim-

itives, it succeeds in 94.67%. Note that the network was not trained with these geometries—we would expect even higher performance if these meshes were included in the training set.

MπNets is Not Suitable for Unsolvable Problems To evaluate performance on unsolvable problems, we generated a set of 800 planning problems in randomized tabletops where the target is in collision with the table or an object on the table. When used for these problems, MπNets showed a 64.25% collision rate.

MπNets is Not Improved by Combining Experts We trained MπNets-C on a combination of 3.27M demonstrations each from the *Hybrid Planner* and *Global Planner*. Environments may have overlapped in these data sets, but entire problems, *i.e.*environment, start, and goal, did not. In problems solvable by the global planner, MπNets-C—like MπNets-G—outperformed MπNets-H in terms of target convergence (97.17% vs 87.72%). While its collision rate is lower than MπNets-G, (18.56% vs 21.94%) MπNets-C's collision rate is still significantly higher than MπNets-H (11%). The behavior of MπNets-C is essentially an average of MπNets-G and MπNets-H, which we attribute to the lack of easily learnable obstacle avoidance behavior by the *Global Planner*. These demonstrations equate to additional noise in the training data, which creates less successful obstacle avoidance behavior. In future work, we intend to explore how to robustly combine experts for improved performance.

3.6 DYNAMIC ENVIRONMENTS

MπNets is an instantaneous policy that assumes a static world at the time of inference. If the scene changes between inference steps, the policy will react accordingly. If the environment is continually changing–as is often the case in dynamic settings–MπNets implicitly approximates the dynamic movement as a sequence of static motions. When the scene changes are slow, this assumption works well. When the changes are fast, it does not. To demonstrate this, we evaluated MπNets in a static tabletop environment with a single, moving block placed on the table. We generated 1000 planning problems across

the table with the block placed at different locations. We specifically chose problems where MπNets succeeds when the block is stationary. When moving, the block follows a periodic curve in x and y, but the two curves have indivisible periods, preventing repetitive movement. We then moved the block at three different speeds: slow, medium, and fast and measured the success rate. At these speeds, MπNets succeeds 88.1%, 57.4%, and 28.3% respectively.

3.7 REAL-WORLD EXPERIMENTS

3.7.1 *Real Robot Evaluation*

We deployed MπNets on a 7-DOF Franka Emika Panda robot with an extrinsically calibrated Intel Realsense L515 RGB-D camera mounted next to it. Depth measurements belonging to the robot are removed and re-inserted using a 3D model of the robot before inference with MπNets. We created qualitative open-loop demonstrations in static environments and closed-loop demonstrations in dynamic ones. Rollouts are between 2 and 80 time steps long depending on the control loop frequency. We demonstrated MπNets in a variety of tabletop problems using a Franka Emika Panda 7-DOF manipulator. A calibrated Intel Realsense L515 RGB-D camera is placed in front of the robot's workspace, viewing the table and potential obstacles on top of it. Point cloud measurements are filtered to remove all points belonging to the robot geometry. The remaining cloud is downsampled to 4096 points and treated as the obstacle. The filtering process runs at 9 Hz. We investigated two different control methods:

Results can be viewed at https://mpinets.github.io and the attached video. As can be seen, MπNets can achieve *sim2real* transfer on noisy real-world point clouds in unknown and changing scenes.

OPEN-LOOP MOTION: Using a fixed, user-defined goal location and the current depth observation, MπNets is rolled out over 80 timesteps or until goal convergence. The resulting path is used to compute a time-parametrized trajectory [83] which is then tracked by a po-

sition controller. The videos listed under "Open Loop Demonstrations" at https://mpinets.github.io show a mix of sequential motions toward pre-defined goals. In some of the examples, the objects are static throughout the video and in others, we re-arrange the objects throughout the video. Despite the changing scene, these are still open-loop demos. While the motions adapt to changing obstacles in the scene, the policy only considers scene changes that happen before the execution of a trajectory. This is because the point cloud observations are only updated once the robot reaches its previous target.

CLOSED-LOOP MOTION: To account for dynamic obstacles MπNets is rolled out for a single timestep at the same frequency as the point cloud filter operates (9 Hz). A time-parametrized trajectory is generated by linearly interpolating $\approx$ 70% of the rolled out path. As in the open-loop case, the resulting trajectory is tracked by a PD controller at 1 kHz. The videos listed under "Closed Loop, Dynamic Scene Demonstrations" at https://mpinets.github.io show examples of boxes thrown into the robot's path while it is moving towards a user-defined target. The evasive maneuver shows MπNets' ability to react to dynamic obstacles.

3.8 LIMITATIONS

While MπNets can handle a large class of problems, they are ultimately limited by the quality of the expert supervisor and its need for a large, diverse dataset of training examples. Both generating the data and training MπNets is computationally intensive, requiring access to equipment that is both economically and environmentally expensive. It will also struggle to generalize to out-of-distribution settings typical of any supervised learning approach.

TRAINING DISTRIBUTION The limitations of the *Hybrid Planner* translate to limitations in the trained policy network. Certain target poses and starting configurations can create unanticipated behavior. When target poses are narrowly out of distribution, the rollout fails

to converge to the target, but as a target poses drifts further from the training distribution, behavior becomes erratic. Likewise, random, initial configurations–such as from rejection-sampling based inverse kinematics–can create unexpected behavior, but we did not observe this in our real robot trials running the policy continuously to a sequence of points. With an improved expert, *e.g.*one with the consistency of our *Hybrid Expert* and guaranteed convergence of the *Global Planner*, we anticipate that the occurrence of failure cases will diminish. We also do not expect the network to generalize to wholly unseen geometries without more training data. But, in future work, we aim to improve the generalization of this method with more data, much in the way that Large Language Models [127] continue to improve generalization, as well as by employing strategies to address covariate shift such as DAgger [139] and domain adaptation.

REAL ROBOT SYSTEM When used on a real robot, performance will degrade as the robot's physical environment drifts from the training distribution. Likewise, performance will degrade with increasing point cloud noise. In order to ensure safe operation in a real-robot system, MπNets could be combined with a collision checker–either one with ground-truth or a learned, such as Scene Collision Net [36]. The collision checker could be used to a) stop the robot before hitting collisions b) make small perturbations to nudge the policy back into distribution or c) enable a traditional planner to plan to the goal. Alternatively, a safety component could be trained to detect whether the scene is outside of the training distribution to alert the operator that MπNets is ill-equipped to handle a particular scene.

In a physical system, not all problems will have feasible solutions and MπNets will often collide in these scenarios, underscoring the need for some additional safety mechanisms to prevent catastrophic behavior. Additionally, MπNets has no concept of history and can collide with the scene if, for example, the robot arm blocks the camera mid-trajectory. To mitigate this, the perception system could employ a historical buffer or filter to maintain some memory of the scene.

EMERGENT BEHAVIOR In some of our test problems, we observed that MπNets produces a rollout where the final gripper orientation is 180° off from the target about the gripper's central axis (*i.e.*the central axis parallel to the fingers). In the test set of problems solvable by the *Global Planner*, this occurs in 2.44% of rollouts. We suspect this behavior is due to the near-symmetry in the gripper's mesh about this axis. The minor differences between the two sides of the gripper may not provide enough information for the Pointnet++ encoder to distinguish between these two orientations. While the rollout does not match the requested problem, this behavior can be desirable in some circumstances. For example, because grasps are symmetric with the Franka Panda gripper, a 180° rotation is preferable if it reduces the likelihood of a collision. For applications where this behavior is unacceptable, we could replace the target representation in the point cloud with points sampled from a mesh with no symmetry.

3.9 CONCLUSION

MπNets is a class of end-to-end neural policy policies that learn to navigate to pose targets in task space while avoiding obstacles. MπNets show robust, reactive performance on a real robot system using data from a single, static depth camera. We train MπNets with what is, as far as we are aware, the largest existing dataset of end-to-end motion for a robotic manipulator. Our experiments show that when applied to appropriate problems, MπNets are significantly faster than a global motion planner and more capable than prior neural planners and manually designed local control policies. Code and data are publicly available at https://mpinets.github.io.

Part III

IMPROVING SAFETY IN END-TO-END SYSTEMS

While MπNets demonstrated the potential of large scale learning for end-to-end collision-free motion, there were many scenarios in which we still observed collisions. While the expert pipeline we used was an effective demonstrator for many problems, it's conservative nature led many common tasks to be out of distrubution for our dataset. Despite collaborating with the authors of the constituent parts of our expert pipeline, *i.e.*OMPL [156] and Geometric Fabrics [173], there was no straightforward way to improve the expert's generality. And, when we trained MπNets with a less conservative expert, we found it to collide much more often. At the time, we believed that this was due to the inconsistent behavior of a sampling based planner. However, over time, it also became apparent that it was also due to how motion planners will often skate close to obstacles, leaving little margin of error for a trained policy trying to mimic the demonstrator.

In the following chapter, I describe my work in [49] to improve the safety of large scale learning through an improved architecture and a novel fine-tuning technique that uses trajectory optimization to explicitly correct the policy's performance. Together, these techniques lead to significant improvements that bring the collision rate below 2% in a challenging set of test cases.

I will also share ongoing explorations into alternative perceptual representations and action spaces. The goal of these explorations is not to present a thorough study, but to investigate how currently popular ideas in the literature can take a role in training a model for end to end collision free motion.

4

AVOID EVERYTHING: MODEL-FREE COLLISION AVOIDANCE WITH EXPERT-GUIDED FINE-TUNING

4.1 INTRODUCTION

The world is full of clutter. Humans effortlessly navigate through complex, unfamiliar spaces while constantly avoiding hazardous collisions. Robotics has not solved this key challenge, which is critical to real-world success of robotic actors [115]. Avoiding collision is one of the most important considerations in robot safety, and many important robotics settings such as kitchens, factories and warehouses are dynamic, restricted, and cluttered. For robotic arms, this problem is especially pronounced due to their complex kinematics (see Fig. 9).

However, many leading methods for collision avoidance rely on a stable, accurate, and fully observed representation of the robot's workspace. Traditional motion planning approaches [9, 88] attempt to explore free space to find a collision-free trajectory. Some of these techniques have rigorous theoretical guarantees that a plan (if one exists) will almost surely be found. Many strategies exist to find a valid path, often by searching through a predefined graph [58, 92] or by sampling the state space to incrementally build a tree [72, 86, 154]. These approaches can require hundreds or thousands of computationally expensive geometric collision checks based on an occupancy model, although there are many techniques to mitigate this disadvantage, *e.g.*lazy edge evaluation [39, 50] and parallelism [161]. Trajectory optimization seeks to find an ideal trajectory according to a set of objectives, typically using soft constraints defined by a differentiable environment model. While these methods can produce smooth motion quickly [42, 157], they require more information about an environment than typical motion planners, such as surface normals, to produce gradients. Whether these methods are using search, sampling,

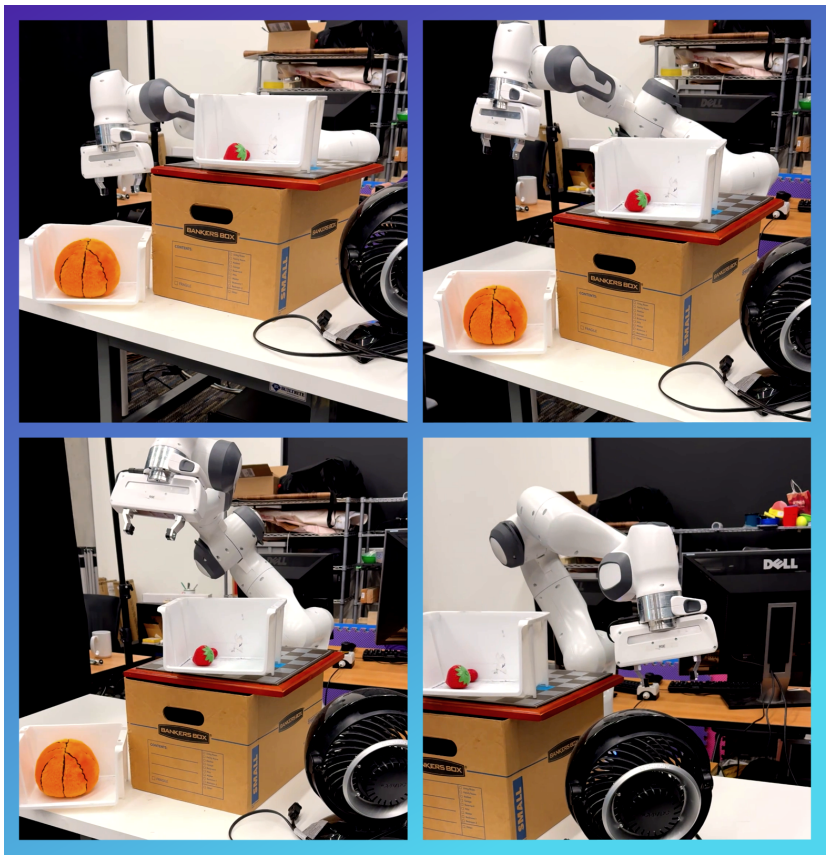


Figure 9: *Avoid Everything* is able to generate collision-free trajectories around complex obstacles in real time, using input from a single depth camera.

or optimization, they require an accurate world model that is either predefined or reconstructed from sensor data. When the world model is inaccurate, the planners may produce unsafe behavior. Furthermore, the real-time performance of these methods can be highly sensitive to the actual distribution of obstacles the robot encounters [30].

Building an accurate world model, particularly in cluttered spaces, is an open problem [52]. End-to-end imitation learning is a popular technique that learns behavior without explicitly modeling the world, instead relying on patterns in how the expert behaves in response to the environment. However, these methods face the challenge of learning to avoid collisions from collision-free demonstrations alone. To address this, traditional motion planners can be used to track the paths produced by the network [19, 148] or directly incorporate a predefined world model into the learning framework [143]. These systems have the same limitations as traditional ones. When the world

model is inaccurate, the system may collide. While expert demonstrations can be made collision-free, learning collision avoidant behavior necessitates a deep understanding of the interplay between the scene geometry and the robot's kinematics. Successful approaches have employed large datasets [47, 107] or explicit losses to encourage obstacle avoidance [47]. However, these techniques still fail in complex problems, leading to constrained capabilities [47] or continued reliance on traditional collision checking techniques [125, 143].

To address these challenges, we present *Avoid Everything*, an end-to-end system that uses point clouds to generate goal-directed, collision-free motion for a robotic manipulator in cluttered 3D scenes. *Avoid Everything* uses a new network architecture *Motion Policy Transformer* (MπFormer) that is trained end-to-end using expert supervision from a motion planner. Our model predicts single-step changes in joint configuration using point cloud observations, the robot's current configuration, and a target end effector pose. We also introduce a fine-tuning approach inspired by hard negative mining [35, 46]: *Refining on Optimized Policy Experts* (*ROPE*). *ROPE* is critical to reducing the collision rate in reaching toward the target. Through experiments, we show that *Avoid Everything* is able to safely solve over 63% of problems where the previous state of the art method [47] fails, resulting in an overall success rate of over 91% in challenging, partially observed manipulation settings. We also demonstrate that *ROPE* can be used as a general tool to reduce collisions, even in conjunction with DAgger [138], a standard technique for improving imitation performance.

Our primary contributions are:

1. We present *Motion Policy Transformer*, a new model architecture designed for predicting goal-directed robot motion from a point cloud and target location.

2. We present *Refining on Optimized Policy Experts*, a novel fine-tuning algorithm for learning collision avoidance in robot motion generation.

3. We demonstrate empirically that *Avoid Everything* reduces the collision rate of the previous state of the art by over 77% and improves success rate by 63%.

4.2 RELATED WORK

REACTIVE CONTROL AND MOTION PLANNING Robot motion generation has traditionally been studied in the context of motion planning with a vast literature of methods [29, 85] based on graph search [58, 90, 92], sampling-based motion planning [16, 50, 68, 73, 86, 88], and *trajectory optimization* [34, 42, 102, 134]. While modern motion planning frameworks can achieve low control latency [117, 146, 157], they assume complete knowledge of the environment and make strong assumptions about obstacle representations for fast collision checking. Perception-driven reactive control of robots also has a rich history. Operational Space Control (OSC) methods such as [8, 24, 130] can enable robots to perform highly dynamic tasks at high control frequencies. However, their myopic nature can lead to local minima in the presence of obstacles. In a similar spirit to our work, Model-Predictive Control (MPC) approaches [14, 67] try to balance reactivity and planning horizon; however, real-time requirements often warrant the use of simple obstacle representations and short horizons that can still lead to local minima.

POINT CLOUD PROCESSING Point clouds, unordered sets of 3D points, are a lightweight and convenient 3D representation. Unlike other 3D representations such as meshes or Signed Distance Functions (SDFs), it is much easier and faster to obtain 3D point clouds from sensors such as depth cameras. As a result, many recent works choose to infer semantics and affordance from point clouds directly, skipping the need for 3D reconstruction. Leveraging powerful neural backbones that process point sets [124, 177], existing networks can segment objects [124], plan grasps [178] and check collisions [109] from partial point clouds only. Following the same spirit, in this work

we show how to reliably generate collision-free joint trajectories from raw 3D point clouds.

IMITATION LEARNING Imitation learning describes a broad class of techniques to learn a policy from demonstrations, often made by a privileged expert [116]. Among imitation learning techniques, behavior cloning [4, 121] describes a set of techniques where a policy is directly trained to mimic an expert's actions. In manipulation, these actions are often phrased as end effector waypoints [18, 19, 113, 148], but these methods require a separate planner and collision checker to perform tasks safely. Recently [10, 181] have demonstrated strong capabilities for using transformers [169] to solve complex manipulation tasks with images as input and joint controls as output. Inspired by these methods, our architecture produces joint space controls given point cloud input.

One common challenge with imitation learning is the problem of covariate shift. Learned policies typically have some small error in prediction. As the error accumulates, the policy will encounter unseen regions of the state space. While many techniques have been proposed to address this issue, a common strategy is to add a wider variety of possible states into the training dataset. This can be done with noise injection—Laskey et al. [89] use a purposefully noisy expert during data collection, while Ke et al. [74] use training-time noise injection to augment previously collected data. It can also be done by fine-tuning a pretrained policy. DAgger [138] uses the pretrained policy to augment the training data by providing expert demonstrations from states visited by the policy. A related technique can be found in computer vision called Hard Negative Mining [35, 46], which augments the training data by labeling the explicit failures (the *hard negatives*) from a pretrained model before either retraining or fine-tuning. Our technique draws inspiration from both DAgger and Hard Negative Mining to explicitly correct the difficult states found from a pretrained model.

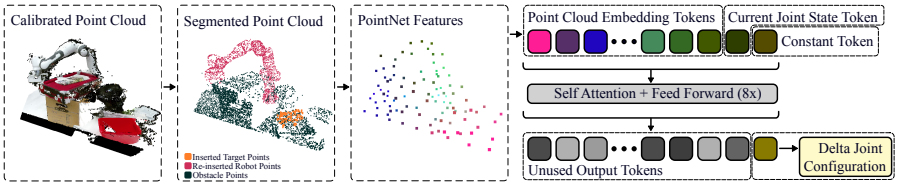


Figure 10: The input to MπFormer is a labeled point cloud, consisting of 4096 points from the depth image (with robot removed), 2048 points sampled from the robot mesh at the current configuration and 128 points from the gripper mesh placed at the desired target. The point cloud is encoded with 3 Set Aggregation [124] layers. The resulting features, along with an encoding of the current joint state and a learned query token, are passed through 8 transformer layers. Finally, the output token that corresponds to the query token is decoded into a delta joint configuration.

LEARNED MOTION PLANNING For the task of motion planning, imitation learning can be used either end-to-end or as a component of a traditional system. Some methods use learning to guide a traditional planner, either through a learned sampler [22, 65, 82, 179] or a learned heuristic function [12, 30]. Other techniques [14, 107] rely on a learned collision model [37]. Motion Planning Networks [125] uses a point cloud neural network to generate waypoints that are then verified with a traditional collision checker. Saha et al. [143] uses a diffusion model to produce plans based on the the SDF representation of the environment. Our neural architecture is most similar to Motion Policy Networks (MπNets) [47], which expects a segmented, calibrated point cloud and produces joint space controls. Despite its strong performance on a variety of problems, MπNets is trained with an expert that is smooth but incapable of reaching close to obstacles. As we discuss in Section 4.5.1.2, when the MπNets architecture is trained and evaluated on more challenging problems (using a more expressive expert), the policy often collides.

4.3 METHODOLOGY

In the following section, we describe our policy architecture, training implementation, and *ROPE*, our fine-tuning strategy that introduces hard negatives and explicit corrections.

4.3.1 *Behavior Cloning for Collision Avoidance*

Avoid Everything is a single-step policy that takes in a point cloud of the scene P_t, a 6-DoF target end effector pose p, and the robot's joint configuration q_t, where t represents the current timestep. The scene point cloud P_t is augmented with points sampled from a mesh of the end effector placed at the target pose p and points sampled from the robot arm at joint configuration q_t, as shown in Fig. 10. This geometric representation of joint state q_t and target pose p has superior performance over a numerical representation [47]. The output of *Avoid Everything* at timestep t is a delta joint configuration Δq_t, which is added to the current joint state q_t to form a position target for the robot to follow.

4.3.1.1 *Architecture*

MπFormer uses PointNet++ [124] to encode the point cloud and a transformer [169] to fuse the point cloud features with a representation of the current joint state. The input point cloud has a feature vector of length 4 for every point. All obstacles are assigned the same feature, all target points are assigned the same feature, and each robot point, which are sampled deterministically from the robot's mesh, is assigned a unique feature to disambiguate points on the arm. Our PointNet++ encoding architecture consists of three Set Aggregation (SA) layers. SA layers are a sparse 3D analog to convolutional layers. Each layer receives a point cloud where each point has a feature and outputs a smaller point cloud by using furthest point sampling to select $\frac{1}{4}$ of the points. Then, each sampled point is used as the center of a ball query. The ball query samples up to 64 points inside the ball and concatenates the ball center's coordinates to each point's feature vector. A four-layer MLP is then run on each point and MaxPool [31] collects the points inside the ball to produce a single feature per ball. The layers' ball queries have radii of 5, 30, and 50 centimeters respectively. Our input point cloud always has 6272 points–4096 obstacle points, 2048 robot points, 128 target points. The downsampled point

cloud after the third set aggregation layer has 98 points. Finally, we add 3D positional encoding to each of these 98 points, similar to [178].

The transformer takes a sequence of tokens as input, consisting of the 98 output features of the third SA layer, a token for the current joint configuration, and a learned constant token, similar to the decoder tokens in [181]. We get the joint angle token by passing the joint angles, which are normalized to be between -1 and 1, through a single linear layer. Our transformer has 8 layers with an embedding dimension of 512 and a feed-forward dimension of 2048. To produce the final output Δq, we take the last token of the output sequence and map it through a single linear layer.

4.3.1.2 *Loss Function*

We train *Avoid Everything* according to the same loss functions as MπNets [47]: a task-space behavior cloning loss to encourage the policy to mimic the expert's behavior in task space, as well as a collision-avoidance loss. These losses are applied on predicted joint states, which are computed by adding the model's output (joint angle deltas) to the input joint angles and clamping the sum at the joint limits.

TASK SPACE LOSS The aim of this loss is to compare the physical positions of the policy's predicted robot joint space configuration and the expert's joint space configuration. For both configurations, we use forward kinematic functions $\phi^{\{i\}}(\cdot)$ to map joint angles of the robot q to 1024 points $x^{\{i\}}$ on the robot's surface, represented in 3D coordinates.

$$L_{BC}(\hat{\Delta}q) = \sum_{i=0}^{1024} \|\hat{x}^i - x^i\|_2 + \|\hat{x}^i - x^i\|_1 \tag{19}$$

Like MπNets, we sum L1 and L2 distances in the loss because the sum penalizes both large and small errors. We use a task space loss following MπNets, which demonstrated it to be more effective when reasoning about collision avoidance as small perturbations along the kinematic chain can lead to large deviations for the end effector.

COLLISION AVOIDANCE LOSS The training data was generated in simulation, giving us access to privileged information unavailable during inference, including a signed-distance representation of the scene. To avoid collisions, we use a hinge-based loss on $D(x)$, the signed distance from a point x on the robot to the nearest surface in the scene. Inspired by motion optimization [42, 48, 134], this loss effectively pushes the robot out of regions of collision. As in Equation 19, we use 1024 points $x^{\{i\}}$ on the robot's surface to measure collision.

$$
\begin{aligned}
L_{\text{collision}} &= \sum_i \|h(\hat{x}^i)\|_2, \text{ where} \\
h(\hat{x}^i_{t+1}) &= \begin{cases} -D(\hat{x}^i), & \text{if } D(\hat{x}^i) \leqslant 0 \\ 0, & \text{if } D(\hat{x}^i) > 0 \end{cases}
\end{aligned}
\tag{20}
$$

4.3.1.3 *Training Implementation*

Avoid Everything was trained on an NVIDIA 4090 in batches of 50 using AdamW [93] with a learning rate of 5e−5 and a linear warmup of 5000 steps from 1e−5. On the cubby environment, the model was trained for 1.2 million steps, which took approximately four days.

During training, we add small amounts of random noise to the input configurations, which [74] showed leads to improved robustness. Like MπNets, the training scenes are constructed from primitives, so point clouds can be generated on the fly during training by sampling points from the surfaces of these primitives. Robot points are sampled deterministically from the mesh of the robot. When *Avoid Everything* runs on the real robot, we mask out the robot points in the depth cloud and re-insert them using the same deterministically sampled points from training.

4.3.2 *Expert-Guided Fine-Tuning*

After pretraining on a large dataset of expert state-action pairs, we observe that the policy is highly capable of reaching the target pose. Despite the reaching success, however, it still collides with objects in a

Algorithmus 1 : Refining on Optimized Policy Experts

Result : π

$\pi \leftarrow \pi_{\text{pretrained}}$
$b \leftarrow$ Batch Size
$r \leftarrow$ Correction Ratio
D_{expert} ▷ Dataset containing expert demos
$B_{coll} \leftarrow \{\}$ ▷ Collision correction demos
$B_{free} \leftarrow \{\}$ ▷ Collision-free expert demos
for {state, next_state, tgt, scene} **in** D_{expert} **do**
    $s \leftarrow$ state
    for $j \leftarrow 1$ **to** N **do**
        $s' \leftarrow \pi(s, tgt)$
        ▷ If s' collides, correct & add to buffer
        if COLLIDES(s', scene) **then**
            $\bar{s}' \leftarrow$ CORRECT(s', scene) ▷ Eq. 20
            ADD(B_{coll}, {$s, \bar{s}', tgt$, scene})
            break
        end
        ▷ If rollout finishes w.o. collision, add orig. example to buffer
        if REACHED(s', tgt) **or** $j = N$ **then**
            ADD(B_{free}, {state, next_state, tgt, scene})
            break
        end
        $s \leftarrow s'$
    end
    if $|B_{coll}| > rb$ **and** $|B_{free}| > (1-r)b$ **then**
        ▷ Make batch & clear buffers
        $B \leftarrow \{\text{POP}(B_{coll}, rb), \text{POP}(B_{free}, (1-r)b)\}$ ▷ Compute loss, perform gradient update
        $\pi \leftarrow$ UPDATE(π, B)
    end
    ▷ Reached validation accuracy or timeout
    if TERMINATION_CONDITION(π) **then**
        terminate
    end
end

significant percentage of problems in the held-out validation set (see Section 4.5.1.1). When we roll out the pretrained policy in simulation, we observe that the first obstacle penetrations are typically shallow and can be pulled out of collision by optimizing the configuration with respect to Equation 20. Based on this observation, we introduce a novel technique of refining the pretrained policy for improved collision avoidance using online fine tuning, inspired by Hard Negative Mining [35, 46] and trajectory optimization methods [42, 93, 134]. We outline our method, which we call *Refining on Optimized Policy Experts (ROPE)*, in Algorithm 1. During the refining stage, we take mini-batches of training data—the same data used for pretraining—and roll out trajectories for a fixed horizon. These trajectories can reach the target, collide, or neither. If the trajectory collides, we capture the state preceding the collision as input and optimize the colliding configuration using Equation 20 to use as supervision and store this state-action pair in a buffer. If the trajectory does not collide, whether by successfully reaching the target or hitting the maximum rollout length, we use a separate buffer and store the input and output from the training batch, unmodified. In order to perform a weight update on the policy, we use a modified mini-batch made up of some proportion r of corrected examples and some proportion $1 - r$ of unmodified expert examples. Once there are sufficiently many examples in both buffers, we remove these examples, assemble them into a modified mini-batch, and perform a weight update according to the losses used during pre-training. As discussed in Section 4.5.1.5, increasing r leads to lower collision rate but poor target convergence.

During fine-tuning, we continually use the latest policy to perform rollouts, even as it is updated. We perform our optimization procedure using AdamW for simplicity, although we expect other methods typical to motion optimization such as Gauss-Newton or Levenberg-Marquardt may lead to a faster fine-tuning procedure. In our best-performing fine-tuning experiment, we reached peak performance after 21 hours of training.

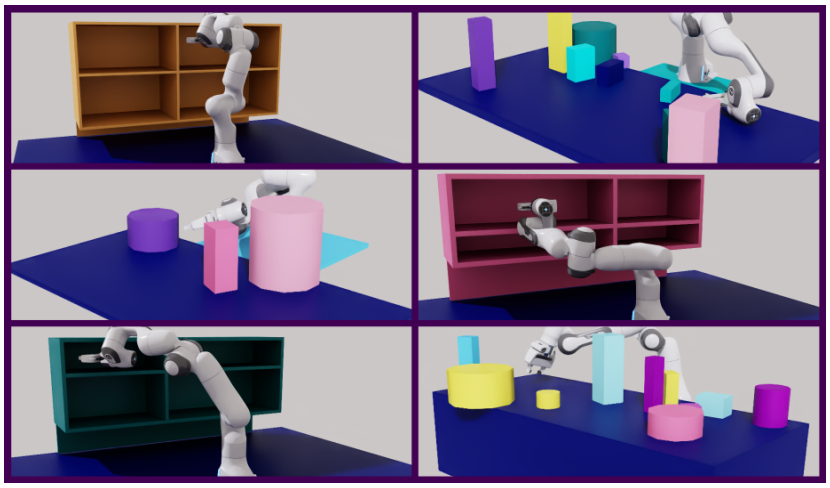


Figure 11: *Avoid Everything* is trained separately in two classes of procedurally generated environments. The first class is a 2x2 cubby with random dimensions and positions. The second is a table of varying dimensions with 3 to 15 objects placed on it. We randomly sample start and goal poses and use inverse kinematics to produce joint configurations to match the poses. Our expert plans are generated in two steps: 1) run AIT* [154] with a configuration space path length objective to find a collision free path; 2) smooth the path with a spline-based shortcutting method [60] and resample the path with a fixed step interval.

4.4 DATA GENERATION PIPELINE

We trained *Avoid Everything* on a large dataset of expert demonstrations in procedurally generated environments, examples of which are shown in Figure 11. The environments themselves were generated randomly and lie within two categories: 2x2 cubbies with randomized dimensions, cubby sizes, and world placement; and tabletops with a collection of randomly placed obstacles. These environments are similar to those demonstrated in MπNets, but they differ in two key ways: we augmented the cubby design to encourage reasonable expert behavior by adding a floor beneath the robot, and we increased the complexity of the tabletop environment by adding more objects and increasing the range of reachable poses. Within these constructed environments, we randomly sample end effector poses and their corresponding inverse kinematics (IK) solutions, which we compute using IKFast [40]. For the cubby environments, the poses are all grasping positions inside a cubby. For the tabletop, the poses are grasps

pointing toward the lower hemisphere and placed either near the table's surface or on top of the objects. We also add neutral configurations drawn from uniform distribution around the robot's default pose to the tabletop data. These poses, for both types of environments, must be at least 5mm away from obstacles. We then use AIT* [154] with a path-length objective combined with a spline-based shortcutting [60] to generate expert demonstrations. In our planning pipeline, we impose a 20 second time limit in which we sample uniformly from the robot's configuration space, marking any sample that is either in self-collision or within 5mm of an obstacle as invalid. During the smoothing stage, we fit a collision and dynamics-aware spline to the planned path while shortcutting. We then sample from the spline at a fixed timestep, leading to paths with similar velocities, but varying lengths.

We chose this sampling pipeline because it enables us to produce expert demonstrations that lie precariously close to obstacles. Previously, MπNets [47] demonstrated strong performance when trained with a so-called *Hybrid Expert*, which uses a reactive controller [173] to follow a planned end effector path. While this expert is effective for learning, it is highly conservative, preferring to stay far away from obstacles. In their experiments, the authors demonstrated that the *hybrid expert* demonstrations are insufficient to learn to solve problems that lie very close to obstacles. With our sampling expert, we chose a 5mm buffer from obstacles because this is sufficiently close for most tasks. As we designed our expert, we observed that increasing the collision margin improves learned collision avoidance, but this limits the expert's ability (and thus, the policy's ability) to plan to targets near obstacles.

We trained *Avoid Everything* separately on each class of environments. For the cubby model, we used 1.25 million problems across 21,604 environments. For the tabletop model, we used 2 million problems across 43,646 environments. To generate this data, we used a single desktop with a AMD Ryzen Threadripper 3990X 64-Core Processor. Generating the cubby and tabletop data took four and six days respectively.

Table 6: *Avoid Everything* Compared to MπNets

Planner	Environment	
	Cubby	Tabletop
	SR (%) / SCR (%) / RSR (%)	SR (%) / SCR (%) / RSR (%)
Avoid Everything	95.71 / 0.50 / 99.30	91.97 / 1.03 / 98.44
MπFormer w. *ROPE*	92.78 / 2.37 / 98.41	89.57 / 4.15 / 96.82
MπFormer	89.92 / 6.43 / 99.52	86.00 / 11.26 / 99.57
MπNets w. *ROPE*	87.35 / 4.72 / 96.68	88.75 / 3.30 / 95.60
MπNets	79.65 / 15.16 / 99.09	77.95 / 14.72 / 95.69

4.5 EXPERIMENTS

In order to evaluate *Avoid Everything*'s performance, we used a mix of quantitative experiments in simulation and qualitative tests on physical hardware. Our simulated experiments are in environments drawn from the same distribution as our training data. However, there are no shared environments between the evaluation and training problem sets.

4.5.1 *Simulated Experiments*

For the following experiments, we use two different evaluation settings. The first is a set of fully observed scenes where we sampled point clouds directly from the mesh. For these experiments, we used 10,000 problems for each environment type. As discussed in Section 4.4, our models are trained on fully-observed point clouds to allow for fast, on-the-fly data generation. We use these experiments to evaluate model features and fine tuning techniques.

We evaluate *Avoid Everything* against other planning techniques using partially observed point clouds generated with synthetic depth images. While our model was trained in fully-observed settings, our aim is for it to work robustly in partially observed environments. For these evaluations, we used 5000 problems in each of our cubby and tabletop environments (10,000 total). For each environment, we captured a synthetic depth image from a randomly positioned camera facing the scene. While Fishman et al. [47] evaluated Motion Policy

Table 7: Planner Performance in Partially Observed Scenes

Planner	Perception	Environment	
		Cubby	Tabletop
		SR (%) / SCR (%) / RSR (%)	SR (%) / SCR (%) / RSR (%)
Avoid Everything	End-to-end	92.34 / 3.10 / 98.68	87.62 / 4.28 / 97.70
MπNets	End-to-end	80.80 / 13.79 / 99.04	74.08 / 18.17 / 94.66
RRTConnect	Octomap	32.68 / 67.16 / 99.52	46.52 / 53.30 / 99.62
cuRobo	NvBlox	73.06 / 22.88 / 94.74	76.86 / 22.11 / 98.68

Networks in partially observed settings, the camera viewpoint was fixed per class of environment. In order to rigorously evaluate the robustness to partial observability, we used random viewpoints for our synthetic images. For each of these images, we placed the robot at a fixed neutral starting configuration and segmented the robot out of the image. To randomize the camera, it was first placed in the scene at a predefined location facing the robot and obstacles, and was then rotated randomly by up to 30° about the z-axis (rotating side to side), then again by up to 10° about the camera's local x-axis (tilting up and down). Both of these rotations were applied using a fixed pivot point directly in front of the camera. Finally, the camera was translated randomly along the global z axis and y axes by up to 25cm.

METRICS We show the results of these experiments in Tables 6, 7 and 8. Each table reports three metrics in each environment class. *Reaching Success Rate* (RSR) is the percentage of problems for which each method could provide a path (collision-free or not) to within 1cm and 15° of the goal. *Scene Collision Rate* (SCR) is the percentage of these paths that had a collision with the scene. *Success Rate* (SR) is the percentage of problems that had a collision-free solution to the goal, including self-collisions.

4.5.1.1 *MπFormer*

Without fine-tuning, MπFormer succeeds in 89.92% and 86.00% of our cubby and tabletop problems. However, after using DAgger and *ROPE*—the version we label *Avoid Everything* in Table 6—we see it succeed in 95.71% and 91.97% in the cubby and tabletop settings

respectively. As discussed in Section 4.5.1.4, DAgger and *ROPE* improve performance independently, and the combination of the two leads to best-in-class performance.

In partially observed settings, we find that *Avoid Everything* still demonstrates strong performance, albeit with a slight drop. Robustness to perspective changes and incompleteness is a well-understood property of PointNet [122]. As described in Section 4.3.1.1, the PointNet layers use MaxPool [31] to aggregate data across spatial regions. MaxPool selects a single salient feature and will have a similar response given the presence of redundant information, which leads the PointNet to be highly robust to missing points. While this architecture-based adaptation is empirically effective, we expect that performance would improve even more if the network were trained on partial view point clouds. Doing so at this scale, however, would require significant computational resources that we do not currently have available, and so we leave such an effort to future work.

4.5.1.2 *Motion Policy Networks*

Our system design is most similar to MπNets, which is the state of the art for learned end-to-end collision free motion. In order to evaluate our method, we trained MπNets on our expert data and compared it to MπFormer without any fine-tuning. We also fine tuned both models using *ROPE* and compared the performance. These results are shown in Table 6. Without any fine-tuning, we found MπFormer to outperform MπNets in both environments. We attribute this difference to the attention layer in our model, which maintains spatial structure of the point cloud, which MπNets flattens in its decoder. Additionally, we find that *ROPE* significantly improves the performance of both models, reducing collision rates by more than half. However, after running *ROPE* on both algorithms, we find that the reaching success rate degrades more for MπNets through the fine-tuning process. MπFormer is better able to adapt to the hard negative examples without losing the ability to reach the target. We also compare the performance of *Avoid Everything* to MπNets in partially observed settings (Table 7) and observe that *Avoid Everything*'s collision rate is less than

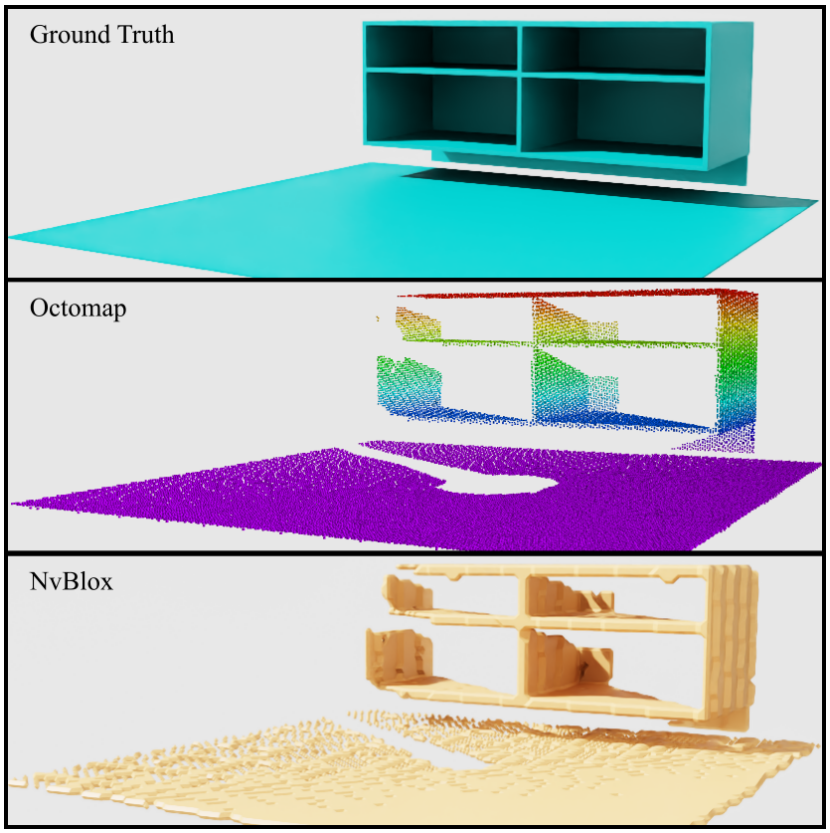


Figure 12: A typical failure case for classical planners is that they do not account for collisions in unobserved regions. In this example, the reconstructions from both Octomap [63] and NvBlox [100] leave large holes due to occlusion. *Avoid Everything* is able to leverage learned priors to produce safe movement without an explicit reconstruction.

$\frac{1}{4}$ of MπNets's collision rate in these settings. Despite its improved success rate, *Avoid Everything* is less suitable for high-frequency control than MπNets. Running on a NVIDIA 4090 GPU, we can run *Avoid Everything* at 33Hz. In the original paper, the authors of MπNets report it running at 150Hz, however we have observed it running at up to 1000Hz on updated hardware.

4.5.1.3 *Classical Methods*

While classical motion planners are highly capable of finding collision-free solutions, some even providing guarantees [56], this hinges on the ability to verify states with a good perceptual model. In practice, we have found that many of the perceptual models used in classical planning lead to erroneous solutions, *i.e.*solutions that the plan-

ner reported as valid when in fact they might have a collision. We evaluated this with two different styles of planners and their perceptual representations–see Figure 12 for examples of the perception. First, we evaluated with the commonly used motion planning library MoveIt! [27] paired with Octomap [63] for perception. We used an Octomap with a resolution of 5mm and RRTConnect [81] (with a 5s timeout) as the planner. In the cubby settings, we found that the planner found a solution in 99.52% of the problems and we attribute the remaining to noise that could be addressed with a longer timeout. However, of these successful plans, over 67% of them had collisions. We attribute this to the randomness in the path due to the sampling procedure, which leads the robot to move unnecessarily in unobserved space. RRTConnect produced fewer collisions (53%) in the tabletop setting, likely due to fewer or smaller holes in the point cloud. We ran a similar test using a trajectory optimization method designed to produce smooth trajectories, cuRobo [157] and NvBlox [100]. This technique finds a path in 94.74% of of cubby problems, but 22.88% of these trajectories have collisions. We set the nvBlox resolution to 1cm for this test after consulting with the authors of cuRobo [157]. While cuRobo also performed better in the tabletop setting, the difference was not as large as RRTConnect (see Table 7). An advantage of these classical methods is that they did not require special tuning or training for either environment. Despite *Avoid Everything* having stronger performance in both environments, we do not expect it to generalize to wholly new settings as classical methods can.

4.5.1.4 *DAgger*

One of the most common techniques for fine-tuning a learned policy is DAgger[138]. DAgger aids in accounting for distribution shift by asking the expert to provide demonstrations at every state the pre-trained policy would visit. Likewise, *ROPE* can be seen as a way to account for distribution shift by correcting the policy when it fails. While DAgger is a generally useful tool for imitation learning, it requires making many costly calls to the expert. In our case, each expert demonstration requires 20 seconds of computation time, which adds

Table 8: *Avoid Everything* Metrics With Varying Refinement Techniques

F.T. Stage 1	F.T. Stage 2	Environment	
		Cubby	Tabletop
		SR (%) / SCR (%) / RSR (%)	SR (%) / SCR (%) / RSR (%)
None		89.92 / 6.43 / 99.52	86.00 / 11.26 / 99.57
ROPE		92.78 / 2.37 / 98.41	89.57 / 4.15 / 96.82
DAgger		93.19 / 4.08 / 99.54	89.17 / 5.59 / 99.31
DAgger	*ROPE*	94.63 / 1.10 / 99.59	91.10 / 2.45 / 98.41
Cons. DAgger		94.88 / 1.28 / 99.16	91.06 / 2.31 / 98.74
Cons. DAgger	*ROPE*	95.71 / 0.50 / 99.30	91.97 / 1.03 / 98.44

up quickly if a demonstration is needed at every state visited by the policy. We implemented two versions of DAgger as comparisons and show the performance in Table 13. In the first version, we ran the pretrained *Avoid Everything* through its entire training data, collected the trajectories with collisions, and requested an expert demonstration at every step leading up the collision. We found that this technique can improve performance, reducing the pretrained collision rate of 6.43% in cubby setting to 4.08%, but it is not better than *ROPE*, which reduces the collision rate to 2.37%. We attribute this to the fact that the DAgger corrections use the same expert, which often veers very close (5mm) to obstacles. To verify this, we tested a second version of DAgger that uses a more conservative expert for corrections–one with a 2cm collision buffer. We label this more conservative technique *Cons. DAgger* in Table 8. As discussed in Section 4.4, this expert is more limited in the problems it can solve, *e.g.*not those that either start or end within 2cm of obstacles. However, we found that this version of DAgger significantly improves collision avoidance without negatively impacting reaching performance, dropping collision rate in the cubby setting to 1.28%. We observe a similar drop in the tabletop setting, bringing pretrained collision rate from 11.26% to 2.31%. Running DAgger, however, is very computationally intensive—collecting DAgger demonstrations for the policy's failures on our training dataset required nearly five days on a desktop with an NVIDIA 3090 GPU and an AMD Ryzen Threadripper 3990X 64-Core Processor.

When used alone, *ROPE* outperformed DAgger with the original 5mm expert in both the cubby and tabletop settings. Meanwhile, fine-

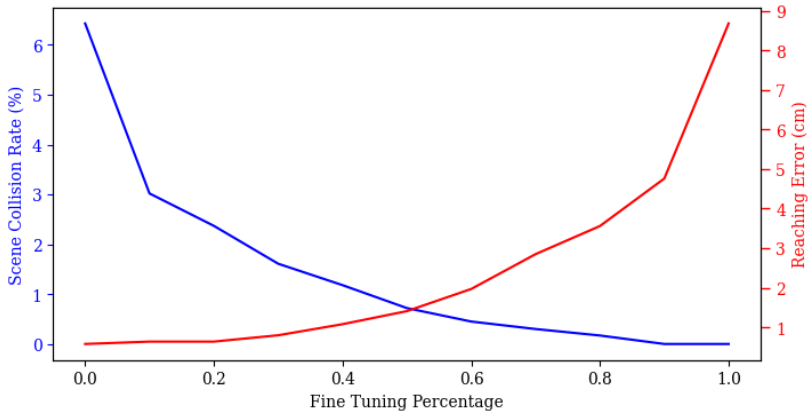


Figure 13: Fine-tuning can be run with different proportions r of hard negative examples. As r increases, the collision rate goes down and target error increases. We attribute this phenomenon to the model overfitting to the hard negatives and forgetting the original behavior cloning objective.

tuning with *Cons. Dagger* outperforms both. However, we did not find *ROPE* to be to be mutually exclusive of DAgger. With both versions of DAgger, we were able to further improve performance by using *ROPE* as a second fine-tuning step. The best performance came from stacking the conservative DAgger technique with *ROPE*, with success rates of 95.71% and 91.97% in the cubby and tabletop settings respectively.

4.5.1.5 *Balancing Collision Avoidance and Success Rate*

We aimed to determine the efficacy of *ROPE* by varying the ratio of hard negative examples in each fine-tuning batch. We set this parameter r as a constant value for the entire fine-tuning procedure and studied how different values change the performance (see Figure 13). For these experiments, we looked only at the cubby setting and used fully observed point clouds, similar to those used during training. We observed a monotonic decrease in collision rate as r increased. However, we also observed a monotonic increase in the reaching error, *i.e.*the minimum distance from the target after rolling out for 70 time steps. With no fine-tuning, we measured an average reaching error of 0.58cm and a collision rate of 6.43%. At $r = 20\%$, we observe an average reaching error of 0.64cm with a collision rate of 2.37%. At $r = 50\%$, collision rate is below 1%, but reaching error averages

1.41cm. We chose $r = 20\%$ for our other experiments, but the choice of this parameter should be determined by the downstream application and the criticality of collision avoidance. We did not experiment with varying r during fine-tuning as a function of performance, but we hypothesize that setting it as a function of performance would improve results.

4.5.2 *Performance on Real Robot Hardware*

We deployed *Avoid Everything* on a physical Franka Emika Panda robot using point clouds from a calibrated depth camera. We used a dual-computer setup running ROS. The control computer, which runs a real-time linux kernel, has Intel(R) Core(TM) i7-4770 CPU with 16 Gigabytes of RAM. The second computer, which runs *Avoid Everything*, has an Intel(R) Core(TM) i9-9900K CPU, 32 Gigabytes of RAM, and an NVIDIA Titan RTX GPU. We use a Kinect V2 for perception, which captures point clouds at approximately 10Hz. We use [44] for eye-on-hand calibration and [15] to remove the robot from the depth cloud; we then re-insert these robot points into the cloud using the deterministic sampling method described in Section 4.3.1.3. We are able to run the model at approximately 25Hz on our hardware, which allows for reactive motion. We send each predicted action directly to a lower level joint controller [11].

The model is able to react to moving obstacles in the scene, but due to speed of our camera, it can take up to 140ms—100ms for the camera update, 40ms for the model update—for the robot to react to an obstacle. We expect that this reactivity could be improved with a faster camera, a faster GPU, or both. We used our best performing checkpoint, which was first fine-tuned with the conservative DAgger pipeline and then fine-tuned with *ROPE* (see Section 4.5.1.4). We observed that the model is excellent at avoiding obstacles on the table when those obstacles are at least partially observable by the camera. We commonly saw collisions into obstacles that were fully occluded or out of the camera's field of view. We expect this issue could be improved with additional cameras to obtain a more complete point

cloud. Many of the obstacles placed in front of the robot were far outside the training distribution, yet the model was able to avoid them easily. However, we found that highly complex obstacles, particularly those with thin structures (e.g. an office chair on its side) can result in collision. Not only was this obstacle out of distribution, but the rear legs were unobserved by the camera, leading to a compounding of our two main challenges.

We found signs of generalization as well as challenges with distribution shift. When we placed the target inside an obstacle, we observed that the model tends to hover above the obstacle without attempting to go in. This is despite the fact that none of the targets in training were ever inside obstacles. However, while this behavior occurred in the majority of cases, the robot did sometimes try to push through an obstacle to reach a target, especially when the obstructing face of the obstacle was obscured from the camera. For example, the top side of a tall box might not be visible by the Kinect, which leads the robot to attempt to push through the top to reach a target placed inside the box. Additionally, the gripper of the Franka is nearly symmetric about the axis that points from the wrist to the midpoint of the fingers. Our training data consisted of randomly generated poses, but these poses typically sampled from only half of the rotations about this axis. When we provided an out-of-distribution pose where the 180° rotation about this axis would be in distribution, we observed the robot typically tries to exploit the symmetry of the gripper and reach the symmetric in-distribution pose. Depending on the application, these 180° rotations may or may not be acceptable. We believe this could be fixed by increasing the variation of target poses in the training set, adding a unique per-point embedding to the gripper points to distinguish orientations, or both.

4.6 LIMITATIONS

While *Avoid Everything* can be trained to have extremely low collisions in complex environments, there are open challenges. First and foremost is the problem of generalization. *Avoid Everything* performs well

for in-distribution tasks, but we do not expect it to perform well in obstacle configurations that are far beyond anything seen during training or fine tuning. Likewise, we would expect a high reaching error for target poses that lie well beyond the training distribution. Second, we used a simple, gradient-based optimization to provide corrections during fine-tuning. For particularly complex environments, providing adequate corrections may require more sophisticated techniques, *e.g.*those used in [157]. Additionally, like other black box learned systems [47], *Avoid Everything* provides no guarantees. Future work could combine *Avoid Everything* with a traditional planner in fully observed settings, similar to [125]. Finally, *Avoid Everything* requires a significant amount of data and compute to train, which can be expensive and environmentally harmful.

4.7 CONCLUSION

Avoid Everything is an end-to-end system that can create safe, collision-free motion toward a goal using only a partially observed point cloud. The system consists of two novel components, MπFormer and *ROPE*. MπFormer is an end-to-end transformer architecture that produces joint space controls toward a target. With no fine-tuning, MπFormer is significantly better than MπNets, the existing state of the art method for end-to-end collision-avoidant motion generation. *ROPE* is a fine-tuning technique used to improve performance by leveraging optimization to correct states where the pretrained policy collides. While we find that *ROPE* can be used to substantially improve the performance of MπFormer, we also found that it can be used to improve MπNets as well. When MπFormer and *ROPE* are used together as *Avoid Everything*, we find that the result is markedly more capable at generating collision-free reaching motion to a goal in partially observed settings than other techniques.

5

ONGOING AND FUTURE WORK

In this chapter, I will discuss ongoing and future experiments that have not yet been submitted for publication. In the last few years, large vision-language models (VLMs) have become increasingly popular and powerful. Seeing the capability of these models, many robotics researchers have applied similar techniques to solve key challenges. As these large VLMs are trained with supervised learning, there is an increased focus on imitation learning for robotics, especially behavior cloning [19, 25, 66, 113, 148]. While there have been many attempts to train general purpose robotic agents, both large and small, none of these methods are yet suitable for general-purpose deployment. Many of these techniques employ standard transformer architectures but present novel design choices for both the observation and action spaces. While these methods have been trained on datasets that range in size, none–as far as we know–have been trained on datasets similar in size to the one presented in MπNets. And, while the tasks presented are often physically impressive, there is no proper way to quantify the relative difficulty of the tasks, other than the learning performance of a particular method. Given the limitations of this data, it can be hard to understand if the trained policies have overfit to the data.

Most of my work in my PhD, *i.e.* MπNets and *Avoid Everything*, seeks to solve a simpler problem–that of collision free reaching. This problem is one of geometry, not physics, which allows for relatively inexpensive data generation. As such, the techniques I've presented can scale to many novel environments and problems. The purpose of this chapter is not to present wholly new ideas, but rather to explore how these popular techniques in the imitation literature scale to our datasets and whether they can be applied to the problem of end-to-end collision free reaching.

Table 9: Preliminary Experiments on Action Space Representations

Action Space	Success (%)	Scene Collision (%)	Reaching Success (%)
Joint Angle Deltas	82.96	9.45	99.99
Absolute Joint Angles	84.62	11.33	98.92
Link Poses w/ *cuRobo [157]*	82.15	11.45	99.51

In the following sections, I will discuss these preliminary experiments. First, I will discuss the new data architecture we used. Then, I will discuss our experiments on several potential action spaces. Finally, I will discuss our work to compare varying perceptual inputs.

5.1 EXPERIMENT DATA

For the following experiments, we used a new dataset of tabletop environments. The purpose of this dataset is to be more similar to a tabletop grasping scenario. As such, this environment has many tall, thin boxes that can be grasped by the robot with a top down grasp. The dataset, and likewise the validation set, consist of problems–and their solutions–reaching from box to box. All starting and target end effector positions are in grasp poses around the box. Ensuring collision free grasps in this environment is challenging because the robot has to successfully move its fingers to the grasping position without clipping the box on the way. See Figure 14 for example renderings of these environments.

5.2 JOINT SPACE ACTION REPRESENTATION

Many robot manipulators are physically controlled by motors in each joint, which makes joint space a logical representation of actions. Within joint space, actions can be represented at varying levels of abstraction from the joints. Low level controllers might directly instruct the motor's torque, whereas higher level controllers might use acceleration, velocity, position, or a combination of these. Other high level controllers such as cartesian-space control are also popular, but these require extra processing as discussed in the next section.

In MπNets, the network predicted joint angle deltas, which, when run at a fixed frequency, are equivalent to velocity control. However, when applying these actions, we integrated the delta and used position-based control for the robot. Position based control is effective in quasistatic environments where the objective is to have the robot track a particular trajectory. While the network predicted deltas that are similar to velocities, we did not ensure that the velocities induced by the deltas respected the constraints of the robot's dynamics. By integrating, we allowed the lower-level position controller to handle the dynamic limits of the system.

Instead of predicting deltas and integrating, the network could instead be tasked with predicting absolute positions, similar to how ACT [181] represents actions. By predicting absolute states, the policy is less subject to compounding integration error over time. However, neural networks are best at predicting zero-centered distributions[61], whereas the distribution of absolute positions generated by a motion planner is much closer to a uniform distribution. However, for any specific task, an almost-surely asymptotically optimal planner will converge on a narrow distribution of possible state. In light of this, we use a similar normalization scheme to the authors of ACT [181] where we assume that the joint angles lie along a multi-dimensional Gaussian distribution. During training, we normalize the states to have zero-mean and unit variance, and when the network predicts actions during inference, the actions are unnormalized according to the mean and variance of the training dataset. Notably, this assumes that there is no covariance between the joint angles–an assumption that holds for sampling based planners with a uniform sampling distribution such as the *Global Expert* used in æbut may not hold for other expert demonstrators, such as Inverse Kinematic Following [6].

These experiments were evaluated with fully observed point clouds. While in previous work (both MπNets and æ), we observed that the PointNet++[124] generalized well to partially observed point clouds, we do not see similar transfer for datasets of this size. Since these are preliminary experiments, we used a smaller dataset, but for a more complete and definitive result, we expect the network to re-

quire a larger dataset of demonstrations. See Table 9 for results. When trained with a sufficient amount of data, we found the performance difference between joint deltas and absolute angles to be small, but meaningful. While the baseline delta joint angle model (without any fine tuning) has a collision rate of 9.45% and success rate of 82.96%, the model trained to predict absolute angles has a collision rate of 11.33% and a success rate of 84.62%. The increased success rate of the absolute angles can be attributed to a significantly lower rate of self collisions. For delta joint angles, the best-performing checkpoint for self-collisions has a rate of 5.1%, whereas the best-performing absolute joint angle version has a self-collision rate of 1.1%. My hypothesis is that the delta joint angle model learns a distribution of valid motions, which prevents the robot from crashing into obstacles, whereas the absolute angle model learns a distribution of valid states, which prevents the robot from crashing into itself. Ultimately, the choice between these action spaces likely depends on the task. For quasistatic tasks with slow movement, avoiding self-collisions may be more important than environment collisions. In dynamic settings with higher velocity movement of both the robot and the scene, environment collisions become more risky. Despite these differences, future models may not require making this choice during training. Instead, it should be possible to co-train with both action spaces. The user could either specify the action representation explicitly or the model could infer on its own based on both the environment and task.

5.3 ROBOT CONTROL VIA CARTESIAN LINK POSE DISPLACEMENTS

Although joint space may correspond to the way many robotic manipulators are physically controlled, it does not correspond well to how most tasks are performed in the physical world. Objects in the world are naturally expressed in terms of 3D poses. As such, any joint space action representation requires additional machinery to map joint angles to cartesian coordinates. The mapping from joint angles to link poses is called *forward kinematics*, whereas the mapping backwards is *inverse kinematics*. While MπNets was able to learn these kinematic

mappings, there are still potential advantages to controlling the robot in cartesian space. Recent works such as [18, 19, 25, 113, 148] demonstrated effective performance for complex manipulation tasks by predicting end effector deltas in cartesian space.

Many manipulation tasks, *e.g.*pick-and-place only require reasoning about end effector movement. And, in simple environments such as those often shown in manipulation skill learning, the robot can use very simple path-planning algorithms without taking collision avoidance into consideration. For example, [25] demonstrated impressive task performance by predicting end effector deltas and using the off-the-shelf inverse kinematic controller that comes with the UR5 robot. If these tasks were to be performed in more cluttered scenes, the system would require a motion planner under-the-hood with collision avoidance, as was used by [148], but these planners then require the typical assumptions made by traditional planning pipelines that are discussed extensively in the previous chapters.

Performing these tasks amid clutter without assuming access to an external collision checker would require reasoning about the entire body. While a joint-space policy can perform dexterous skills [181], given the efficacy of cartesian space policies, we want to understand the feasibility of using a policy that controls the entire body in cartesian space. We hypothesize that there could be several benefits to such a policy. First, a cartesian space policy does not require any specific understanding of a robot's kinematic structure and therefore may be able to learn actions that are solely dependent on the robot's geometry. Consequently, such a policy may be able to control robots with novel morphology, although this assumes some method to project these actions back into joint space. Second, this action space may be more effective for an RGB-based policy because actions can be expressed in the camera frame. With extrinsics, point clouds can be transformed into a canonical view, which means that when the robot's base is fixed, the actions can always be expressed in a constant frame relative to the perception. Joint space actions have a one-to-one mapping with world-frame link poses, which makes them appropriate for perception that can be rendered in a canonical frame, either by

fixing the camera as in [181] or by transforming the perception as we did in MπNets. Meanwhile, the techniques that predict end effector deltas primarily use perception that is either always captured from the same camera viewpoint [19, 148] or by framing actions in terms of the camera [26, 66]

ARCHITECTURE In order to adapt *Avoid Everything* to a policy with cartesian-space actions, we changed the network's output space. In *Avoid Everything*, the learned constant token is decoded via a single-layer MLP into a 7-DOF action that can be integrated to get a joint position command. Assuming that both the base and the fingers are fixed, the Franka Panda robot has 8 movable links. To predict the 8 poses, we use a single layer multi-layer perceptron to predict 72 values, which numerically represent the 8 poses. Each pose is made up on 9 numbers. The first three are the cartesian displacement and the latter 6 express a 3D orientation in the Ortho6D representation [183]. These poses represent a transformation in the robot's base frame ${}^{\hat{base}}T_{base}$ that can be left-multiplied by the current link's pose ${}^{world}T_{link}$ to get the new world frame pose of the link ${}^{\hat{base}}T_{link}$.

PROJECTING TO THE KINEMATIC CHAIN Our network predicts poses without directly considering the robot's kinematic chain. Still, the robot must be controlled via joint space controls, necessitating a conversion from the predicted poses into valid configurations. While there is a one-to-one mapping between the joint angles $\{\theta_i\}^i$ and fully-specified set of link poses for a manipulator $\{\hat{base}T_{link}\}^{link}$, this mapping is not *onto* because there exists many sets of poses that do not correspond to valid kinematic configuration. The forward kinematic mapping from joint angles to link poses is therefore not bijective and cannot be inverted. To find a valid configuration, we instead use nonlinear optimization to find a best-fit configuration to the network's output by solving the following equation:

$$\arg\min_{q} \sum_{j} C(\hat{X}_j, \phi_j(q)) \tag{21}$$

where $\hat{X}_j$ is the predicted pose for link j from the network and $\{\phi_j\}^j$ are the forward kinematic mapping for each link. The cost function $C(\cdot)$ refers to a cost on both poses, which constitutes a cost on both the position distance $\|\hat{p}_j - p_j\|_2$ and quaternion distance $\hat{q}_j^T q_j$ of the poses.

$$C(\hat{X}_j, X_j) = \alpha_0 \log\cosh(\alpha_2 \|\hat{p}_j - p_j\|_2) + \alpha_1 \log\cosh(\alpha_3(\hat{q}_j^T q_j)) \quad (22)$$

RESULTS AND DISCUSSION As in Section 5.2, the reader should note that these evaluations were run with fully observable point clouds. We performed fully observed evaluations on these point clouds because the training data set of demonstration trajectories was not large enough to exhibit the generalization to partial view point clouds that we observed in both MπNets and *Avoid Everything*. During inference, we use CuRobo [157] to solve Equation 21 at every timestep where we use the previous timestep's configuration as the seed. This leads to very smooth motion and a collision rate that is higher, but similar to that of *Avoid Everything*. These preliminary results are shown in Table 9.

Whereas *Avoid Everything* has collision rate of 9.45%, the link pose action space has 11.44%. Meanwhile, the success rate of *Avoid Everything* is 82.96% whereas this method has success rate of 82.15%. It is important to note that this optimization naively tries to find the best-fit inverse kinematic solution, but we could instead employ a traditional 3D reconstruction pipeline such as NvBlox [100] to ensure collision free solutions in observable regions. CuRobo typically solves inverse kinematics (IK) problems, even those required to be collision-free, in under 10 milliseconds, so this method would still be effective for real-time control. A common critique of learned motion generation is that there is no consistent way to prevent catastrophic failure; the proposed IK approach addresses this issue by leveraging the optimizer to reliably avoid collisions and ensure safety around visible obstacles while relying on the network's learned experience to safely guide the robot through unobserved regions. Moving forward, we

Table 10: Preliminary Experiments on Observation Space

Perceptual Input	Success (%)	Scene Collision (%)	Reaching Success (%)
Point Cloud			
40 K trajectories	60.16	34.29	99.90
80 K trajectories	63.22	30.78	99.94
Images			
40K trajectories	23.55	45.05	54.76
80K trajectories	40.60	43.77	73.89

are optimistic that this action space will improve safety and allow us to relax our system requirements on robot morphology and camera calibration.

5.4 IMAGE-BASED COLLISION AVOIDANCE

During my PhD research, I primarily used point clouds as the perceptual representation for our policies. At the time that we started working on MπNets, other robot learning papers, *e.g.*[103], had recently shown the efficacy of point clouds for 3D robotics tasks. When working with geometric tasks, point clouds are a natural representation because there is little sim-to-real gap. With a dataset of high quality meshes, point clouds sampled from these meshes–either via a synthetic depth camera or naive surface sampling–tend to accurately reflect real world point clouds. Additionally, point cloud processing networks are highly robust to missing regions, *e.g.*due to to occlusions in a depth image, and noise, *e.g.*from a low cost sensor. For the problem of end-to-end collision free reaching, the network need only reason about the scene's geometry, which makes point clouds an excellent representation. Additionally, MπNets's scale was vastly larger than anything else at the time, which made rendering performance extremely important. When rendering fully-observed point clouds in simulation, there are many tricks to make point cloud sampling fast including using primitive objects with fast analytic samplers or pre-sampling on a set of possible meshes used in the scene. These tricks were essential to train MπNets in a reasonable amount of time. We were able to generate point clouds on the fly during training, which

improved robustness while also reducing the infrastructure overhead of the system..

In recent years, RGB-image-based policies are increasingly popular [18, 19, 113, 181]. While point clouds accurately capture scene geometry, RGB images are more semantically meaningful. Additionally, RGB sensors are cheap, fast, and work in many situations where depth cameras do not. For example, depth cameras that employ infrared light do not work well in the sun. Depth from stereo is more robust in sunlight but depth accuracy degrades for distant objects where "distant" is determined by the stereo baseline. And, perhaps explicit depth is unnecessary for most tasks. After all, our aim is to train a policy that has human-like performance and humans are highly effective with binocular RGB vision. In this exploration, our goal s to understand whether RGB images, as they are currently used, are sufficient for collision avoidance. In this section, I explore the challenges and benefits of learning a collision-avoidant policy from images alone.

SYNTHETIC IMAGE GENERATION While many policies are trained via monocular images [18, 19, 113], others use calibrated multi-camera setups [181]. Collision avoidance is a precision task that, whether implicit or explicit, requires 3D understanding of the world. As such, we chose to use a multicamera setup to aid the network in understanding the 3D environment. Transferring an image-based policy from a simulated training environment to the real world is very challenging due to limitations in current rendering technology. To circumvent this, image-based policies are typically trained with real world data collected by human demonstration. Large companies are able to amass huge datasets via professional human demonstrators [18], whereas university labs either collectively pool data [33, 77] or rely on low-cost demonstration devices to collect data [26, 176]. For this preliminary study, we avoided the large-scale data collection effort by focusing solely on collision avoidance in simulated environments. To scale this up and deploy it on real robots, however, we expect that

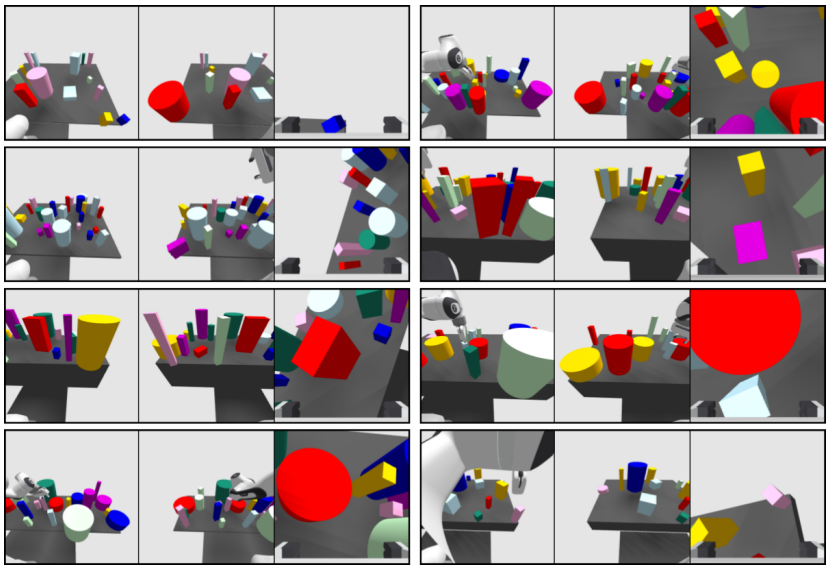


Figure 14: We trained our RGB model using 4.4 million images of synthetic environments. At each state, we randomized the colors of the objects, leading to a unique environment for every state.

would require a significant data collection effort to either supplement or replace the simulated data.

To generate the simulated data, we rendered three images for every state-action pair in the dataset described in Section 5.1. Example images are shown in Figure 14. The dataset consists of many trajectories in each environment, but we randomized the colors of the various primitives shapes in the environments at every timestep to improve robustness [166]. The three camera views consist of two "shoulder" views that are on either side of the robot's base. These views are always captured from a fixed position relative to the base across the entire dataset. The third view is a wrist-mounted camera that faces the fingers, similar to the wrist cameras in [181]. In MπNets and *Avoid Everything*, we used fast geometric samplers to sample point clouds on the fly during training, enabling a highly compact data representation. When rendering images with OpenGL, the rendering is at least an order of magnitude slower. If done on the fly, rendering would make training take many days per epoch. Instead, we pre-rendered the data and stored it. The entire dataset described in Section 5.1 consisted of 318,749 expert demonstrations, which consisted of 5,847,506 states. Rendering the roughly 17.5 million images for this dataset

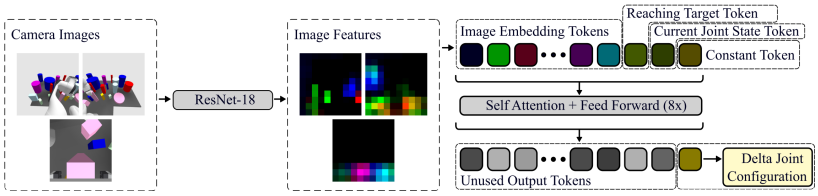


Figure 15: We adapted the MπFormer architecture to process multiview image input by using the image backbone from [181] and feeding the ResNet [61] features into the transformer. The image features in this diagram correspond to ResNet activations. The network appears able to understand the varying components of the scene, but the activations from the wrist camera images do not match those of the other two cameras, suggesting that the ResNet cannot co-identify features between these cameras.

would require 5 terabytes—more hard drive space than we have available in the lab at this time, so this experiment was conducted with a reduced dataset. We tested with a two datasets, one with 40.000 expert demonstrations and one with 80,000 demonstrations, in order to understand whether more data dramatically improves performance. Each tranche of 40,000 demonstrations could be generated on a single NVIDIA 4090 GPU with 10 CPU processes in 14 hours.

ARCHITECTURE The architecture we used for this experiment is an adapted form of MπFormer where we replaced the point cloud encoder with the image backbone architecture used in [181]. This is a ResNet-18 [61] pretrained on ImageNet [141] with fixed parameters for the BatchNorm. While training the policy, we backpropagate through the image backbone to fine tune for our specific task. See Figure 15 for a visual representation.

RESULTS AND DISCUSSION In order to fairly compare the image-based model against the point cloud-based model, we trained both with identically sized datasets. Our aim in this experiment is to make an apples-to-apples comparison between the methodologies. See Table 10 for the results of this experiment. While these results suggest that the point cloud model performs better at this scale, we as-of-yet do not know whether these models will scale in performance at similar rates. There is strong evidence in the literature that the performance of image models scales with model and data size, *e.g.*[78, 175],

whereas to our knowledge, no point cloud models have shown such generalization performance at large scales.

When trained with a single tranche of 40,000 trajectories, we find that the network has a tendency to overfit to the training data. While the training loss is lower than the equivalent baseline loss for MπFormer, the validation loss is significantly higher. This behavior is also reflected during rollouts where the image-based policy demonstrates reasonable behavior, but still has very high collision rates (45.05%). Meanwhile, MπFormer exhibits a collision rate of 34.29. The reaching success rate is also significantly lower for the image-based model—54.76% vs 99.90%. When trained with 80,000 demonstrations, both models improve, but the performance of the image based model is still nowhere near as strong as MπFormer.

The observation that both collision avoidance and reaching success metrics are significantly lower for the image-based model suggests the network is unable to learn an accurate geometric understanding of the environment. Whereas the point cloud policy has explicit access to the scene's geometry—it is directly encoded in the point cloud features—the image-based policy has to implicitly learn 3D representations. This auxiliary requirement adds significant overhead to the end-to-end objective. Furthermore, the cameras each provide very different visual information and the network has no explicit instruction on how to associate images from a shoulder camera with those of a wrist camera. Upon visual inspection, it is clear that the network is able to recognize the robot and obstacles in the scene (see Figure 15), but the activations from the wrist camera do not identify the robot and obstacles in the same way. We hypothesize that training separate image backbones per-camera may improve performance, but this inability to co-identify images may also be due to the pick-and-place nature of the expert demonstrations. These tasks lack significant visual diversity, which makes would make it difficult for the network to co-identify shared features between in the images.

In order for a purely image-based policy to work as effectively as a point cloud model, we hypothesize that we would need to significantly scale up the data, increase the scene complexity, or introduce

auxiliary reconstruction objectives in the form of secondary losses to force the network to pay attention to important regions. However, given that point cloud policies are so effective for geometric tasks, I believe there is no reason to disregard point clouds in favor of images. In a real system, if both point clouds and images are available, we should be using both as inputs to the network. Recent work [113, 114] has demonstrated that co-training on multi-modal inputs can lead to stronger performance, so another potential solution would be co-train with both images and point clouds and use whatever representation is available during inference.

5.5 CONCLUSION

In both MπNets and *Avoid Everything*, we used point clouds as the perceptual input and delta joint angles as the action space. We chose these representations because they accurately capture geometry and provide a natural control interface, respectively. However, recent innovations in the literature have made very different choices for both inputs and outputs. Images capture semantic information about the scene, which can be helpful for downstream tasks, but our preliminary results suggest that they are insufficient for the geometrically precise task of collision avoidance. We also found that delta joint angles remain the best-performing action space, while both absolute angles and 3D poses perform similarly well when trained with enough data.

6

CONCLUSION

In the last decade, there has been a major shift in robotics as the field has progressed from using analytic techniques to purely learned ones. Looking back on my PhD, I was fortunate enough to contribute to three different trajectories of robotics as the academic community made this shift. First, there was the work that inspired me to pursue an academic career. It is rigorous and mathematical, leveraging geometry and probabilistic inference to create robotic behavior. By the time I matriculated, the shift to deep learning was already well underway. While these learned methods showed promise, they tended to live within componentized systems, where the learned component simply replaced a traditional technique. These ideas were particularly effective for the perceptual components of a full-stack robotics system. Traditional perceptual techniques are highly effective for geometric understanding when there is an abundance of visual information—*e.g.*many camera angles—but they are not nearly as effective as neural networks at making a "best-guess" in information-poor settings. Furthermore, neural network systems are far more capable of high level, semantic understanding. Due to this, over the course of my studies, we've seen increasing enthusiasm for an end-to-end learned approach. Outside of robotics, deep learning can already solve incredibly complex problems, and tasks formerly thought impossible for a single neural network, *e.g.*realistic text generation, are now considered more-or-less solvable through scale [21]. The robotics community has been slower to adopt this fully-learned approach, but the field is starting to move in that direction. Whereas in 2020, our idea to solve motion planning with a large network was met with disbelief and near-philosophical disagreement, the reaction in 2024 is quite the opposite. Instead of questions asking us why we would pursue such a crazy idea, now we are told that the idea is so banal as to

have no novelty. This changing perspective is affecting all corners of robotics research and this year, many companies are even springing up to train a large generalized end-to-end model. After witnessing this sea change in the community's perspective, as well as in my own research, the most important lesson I've learned is to pay attention to what is happening in adjacent fields. In hindsight, the enthusiasm for large-scale deep methods in computer vision and language was an obvious sign that these methods would prove to be useful in robotics in much the same way.

While the robotics community collectively pursues the idea of the giant robotics model, the looming question is how we can produce enough high quality data to learn generalizable behavior. This, in fact, has been the big question driving most of my PhD work. Collecting good data is challenging for many reasons: the physicality of a robotic system, the idiosyncrasy of many tasks, and the high dimensionality of the problems. Collecting these large data sets is time consuming and highly capital intensive, which makes it a task better suited for industry. Another important lesson of my PhD has been to pick a version of the problem that is actually achievable with the resources available. In my case, this meant focusing on algorithmic experts and geometric tasks, which allow for automatic data collection. Doing this allowed me to chase generality for the collision-avoidance problem. However, another lesson in my PhD is that collision avoidance is an important but "boring" problem with many existing solutions. While these existing solutions require assumptions, these are assumptions that the community has collectively decided are acceptable. I strongly believe that these assumptions (*e.g.*an accurate collision checker) are inadequate for many real world settings, but I have repeatedly found it difficult to convince the community. And because we have existing systems that work well-enough under commonly accepted solutions, I do not expect this work to see broad adoption until it is a true foundation model that works in any environment. The shift toward end-to-end learning in the community is happening more concretely in the domain of skill learning. However, the skill learning work, *e.g.*[18, 25, 148], has yet to show the same kind of broad gener-

alization performance that we've seen within the MπNets family of work. Moving forward, I expect that the separate ambitions of learning safe full-body motion and complex, dexterous skills will begin to merge. Safe motion planning requires a deep understanding of scene geometry and robot kinematics, something that is also essential for many whole-body skills. Likewise, the semantic information relevant to complex skills informs how a robot moves safely through space when there is no obvious collision-free path.

There has been a lot of discussion, in both academia and industry, about what we still need to create the "GPT moment for robotics." In order for this to happen, I believe that robots must be able to reason safely about their whole bodies in completely novel spaces. While my research has not shown robustness to completely novel settings, we've been able to see fairly strong generalization to in-distribution settings. To make this possible, it was not just the network architecture or the quantity of data—although these things did matter—but also the quality of the data. In MπNets, we tried to naïvely scale up our *Global Expert* dataset, but this did not lead to safe behavior on its own. Instead, we had to focus specifically on what information was missing, which in our case were examples of how to avoid collision in precarious states. I believe that to train a high quality robotics foundation model, we will need a massive dataset of high quality demonstrations, but we do not yet know *how massive* the dataset must be. The last lesson of my PhD has been that we can minimize the requirement to collect such large datasets by looking for gaps in the experience and knowledge represented in a dataset and attempt to correct them. In the next few years, I believe that we will begin to see models that can safely control many types of robots, in a wide range of environments, and with various types of sensors. To train these models, we must scale up our our tasks, our environments, and our demonstrations, but first, the main obstacle ahead is finding the right expert.

BIBLIOGRAPHY

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. "Gpt-4 technical report." In: *arXiv preprint arXiv:2303.08774* (2023).

[2] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Joshua Tobin, P. Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay." In: *NIPS*. 2017.

[3] Haoyu Bai, Shaojun Cai, Nan Ye, David Hsu, and Wee Sun Lee. "Intention-aware online POMDP planning for autonomous driving in a crowd." In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 454–460.

[4] Michael Bain and Claude Sammut. "A Framework for Behavioural Cloning." In: *Machine Intelligence 15*. 1995.

[5] Sivakumar Balasubramanian, Alejandro Melendez-Calderon, Agnès Roby-Brami, and Etienne Burdet. "On the analysis of movement smoothness." In: *Journal of NeuroEngineering and Rehabilitation* 12 (2015).

[6] Aldo Balestrino, Giuseppe De Maria, and Lorenzo Sciavicco. "Robust control of robotic manipulators." In: *IFAC Proceedings Volumes* 17.2 (1984), pp. 2435–2440.

[7] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. "Intention-aware motion planning." In: *Algorithmic foundations of robotics X*. Springer, 2013, pp. 475–491.

[8] C Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter. "Dynamic locomotion and whole-body control for quadrupedal robots." In: *2017 IEEE/RSJ International Conference on Intelligent Robots*

and Systems (IROS). IEEE. 2017, pp. 3359–3365. URL: https://ieeexplore.ieee.org/document/8206174.

[9] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. "Manipulation planning on constraint manifolds." In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 625–632. URL: https://ieeexplore.ieee.org/document/5152399.

[10] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. *RoboAgent: Generalization and Efficiency in Robot Manipulation via Semantic Augmentations and Action Chunking*. 2023. arXiv: 2309.01918 [cs.RO]. URL: https://arxiv.org/abs/2309.01918.

[11] Mohak Bhardwaj. *franka_motion_control*. https://github.com/mohakbhardwaj/franka_motion_control. 2024.

[12] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian A. Scherer. "Learning Heuristic Search via Imitation." In: *Conference on Robot Learning*. 2017. URL: https://arxiv.org/abs/1707.03034.

[13] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D. Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. "STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation." In: (2021).

[14] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation." In: *Conference on Robot Learning*. PMLR. 2022, pp. 750–759. URL: https://arxiv.org/abs/2104.13542.

[15] Nico Blodow. *realtime_urdf_filter*. https://github.com/blodow/realtime_urdf_filter. 2024.

[16] Robert Bohlin and Lydia E Kavraki. "Path planning using lazy PRM." In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia pro-*

ceedings (Cat. No. ooCH37065). Vol. 1. IEEE. 2000, pp. 521–528. URL: https://ieeexplore.ieee.org/document/844107.

[17] Rogerio Bonatti, Cherie Ho, Wenshan Wang, Sanjiban Choudhury, and Sebastian A. Scherer. "Towards a Robust Aerial Cinematography Platform: Localizing and Tracking Moving Targets in Unstructured Environments." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

[18] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Krzysztof Choromanski, Tianli Ding, et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control." In: *ArXiv* abs/2307.15818 (2023). URL: https://arxiv.org/abs/2307.15818.

[19] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, et al. "RT-1: Robotics Transformer for Real-World Control at Scale." In: *ArXiv* abs/2212.06817 (2022). URL: https://arxiv.org/abs/2212.06817.

[20] Tim Brooks et al. "Video generation models as world simulators." In: (2024). URL: https://openai.com/research/video-generation-models-as-world-simulators.

[21] Tom B. Brown et al. "Language Models are Few-Shot Learners." In: *ArXiv* (2020). arXiv: 2005.14165 [cs.CL].

[22] Constantinos Chamzas, Zachary K. Kingston, Carlos Quintero-Peña, Anshumali Shrivastava, and Lydia E. Kavraki. "Learning Sampling Distributions Using Local 3D Workspace Decompositions for Motion Planning in High Dimensions." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 1283–1289.

[23] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. "Differentiable Spatial Planning using Transformers." In: *International Conference on Machine Learning*. 2021.

[24] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. "Rmp flow: A computational graph for automatic motion policy genera-

tion." In: *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer. 2020, pp. 441–457. URL: https://arxiv.org/abs/1811.07049.

[25] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. "Diffusion policy: Visuomotor policy learning via action diffusion." In: *arXiv preprint arXiv:2303.04137* (2023).

[26] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. "Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots." In: *arXiv*. 2024.

[27] Sachin Chitta, Ioan Sucan, and Steve Cousins. "Moveit![ros topics]." In: *IEEE Robotics & Automation Magazine* 19.1 (2012), pp. 18–19.

[28] Jia Pan Chonhyon Park and Dinesh Manocha. "ITOMP: Incremental Trajectory Optimization for Real-time Replanning in Dynamic Environments." In: *International Conference on Automated Planning and Scheduling (ICAPS)*. 2012.

[29] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. English. MIT Press, May 2005. ISBN: 0262033275. URL: https://biorobotics.ri.cmu.edu/book/.

[30] Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. "Data-driven planning via imitation learning." In: *The International Journal of Robotics Research* 37.13-14 (2018), pp. 1632–1672. URL: https://arxiv.org/abs/1711.06391.

[31] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification." In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), pp. 3642–3649. URL: https://arxiv.org/abs/1202.2745.

[32] Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. "Exploring the Limitations of Behavior Cloning for Autonomous Driving." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 9328–9337.

[33] Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, and et. al Ajay Mandlekar. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. https://arxiv.org/abs/2310.08864. 2023.

[34] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. "Whole-body motion planning with centroidal dynamics and full kinematics." In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 295–302. URL: https://ieeexplore.ieee.org/document/7041375.

[35] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection." In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* 1 (2005), 886–893 vol. 1. URL: https://ieeexplore.ieee.org/document/1467360.

[36] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. "Object Rearrangement Using Learned Implicit Collision Functions." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 6010–6017.

[37] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. "Object Rearrangement Using Learned Implicit Collision Functions." In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2021. URL: https://arxiv.org/abs/2011.10726.

[38] Frank Dellaert and Michael Kaess. "Factor Graphs for Robot Perception." In: *Foundations and Trends in Robotics* 6.1-2 (2017), pp. 1–139. ISSN: 1935-8253. DOI: 10.1561/2300000043.

[39] Christopher Dellin and Siddhartha Srinivasa. "A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selec-

tors." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 26. 2016, pp. 459–467. URL: https://arxiv.org/abs/1603.03490.

[40] Rosen Diankov. "Automated Construction of Robotic Manipulation Programs." PhD thesis. Carnegie Mellon University, Robotics Institute, Aug. 2010. URL: http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[41] Jing Dong, Byron Boots, and Frank Dellaert. "Sparse Gaussian Processes for Continuous-Time Trajectory Estimation on Matrix Lie Groups." In: *Arxiv* abs/1705.06020 (2017). URL: http://arxiv.org/abs/1705.06020.

[42] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. "Motion planning as probabilistic inference using Gaussian processes and factor graphs." In: URL: https://www.researchgate.net/publication/304532888_Motion_Planning_as_Probabilistic_Inference_using_Gaussian_Processes_and_Factor_Graphs.

[43] Anca D Dragan, Kenton CT Lee, and Siddhartha S Srinivasa. "Legibility and predictability of robot motion." In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2013, pp. 301–308.

[44] Marco Esposito. *realtime_urdf_filter*. https://github.com/IFL-CAMP/easy_handeye. 2024.

[45] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. "C-Learning: Learning to Achieve Goals via Recursive." In: *International Conference on Learning Representations*. 2021.

[46] Pedro F. Felzenszwalb, Ross B. Girshick, David A. McAllester, and Deva Ramanan. "Object Detection with Discriminatively Trained Part Based Models." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2010), pp. 1627–1645. URL: https://ieeexplore.ieee.org/document/5255236.

[47] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. "Motion Policy Networks."

In: *Proceedings of the 6th Conference on Robot Learning (CoRL)*. 2022. URL: https://arxiv.org/abs/2210.12209.

[48] Adam Fishman, Chris Paxton, Wei Yang, Dieter Fox, Byron Boots, and Nathan D. Ratliff. "Collaborative Interaction Models for Optimized Human-Robot Teamwork." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 11221–11228.

[49] Adam Fishman, Aaron Walsman, Mohak Bhardwaj, Wentao Yuan, Balakumar Sundaralingam, Byron Boots, and Dieter Fox. "Avoid Everything: Model-Free Collision Avoidance with Expert-Guided Fine-Tuning." In: *ICRA Workshop on a Future Roadmap for Sensorimotor Skill Learning for Robot Manipulation*. 2024.

[50] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. "Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search." In: *The International Journal of Robotics Research* 39.5 (2020), pp. 543–567. URL: https://arxiv.org/abs/1707.01888.

[51] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Tim D. Barfoot. "Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), pp. 3067–3074.

[52] Sourav Garg et al. "Semantics for Robotic Mapping, Perception and Interaction: A Survey." In: *Foundations and Trends® in Robotics* 8.1–2 (2020), pp. 1–224. ISSN: 1935-8253. DOI: 10.1561/2300000059. URL: http://dx.doi.org/10.1561/2300000059.

[53] Caelan Reed Garrett. *PyBullet Planning*. https://pypi.org/project/pybullet-planning/. 2018.

[54] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. "Online Replanning in Belief Space for Partially Observable Task and Motion Problems." In: *arXiv preprint arXiv:1911.04577* (2019).

[55] Github. *GitHub copilot · your AI pair programmer*. 2021. URL: https://copilot.github.com/.

[56] Ken Goldberg. "Completeness in robot motion planning." In: *Proceedings of the Workshop on Algorithmic Foundations of Robotics*. WAFR. San Francisco, California, USA: A. K. Peters, Ltd., 1995, 419–429. ISBN: 1568810458. URL: https://goldberg.berkeley.edu/pubs/completeness.pdf.

[57] Elena Corina Grigore, Kerstin Eder, Anthony G Pipe, Chris Melhuish, and Ute Leonards. "Joint action understanding improves robot-to-human object handover." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 4622–4629.

[58] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." In: *IEEE Trans. Syst. Sci. Cybern.* 4 (1968), pp. 100–107.

[59] Kris K. Hauser. "Fast Interpolation and Time-Optimization on Implicit Contact Submanifolds." In: *Robotics: Science and Systems*. 2013.

[60] Kris K. Hauser and Victor Ng-Thow-Hing. "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts." In: *2010 IEEE International Conference on Robotics and Automation* (2010), pp. 2493–2498.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[62] Paul W. Holland and Roy E. Welsch. "Robust regression using iteratively reweighted least-squares." In: *Communications in Statistics - Theory and Methods* 6.9 (1977), pp. 813–827. DOI: 10.1080/03610927708827533. eprint: https://doi.org/10.1080/03610927708827533. URL: https://doi.org/10.1080/03610927708827533.

[63] Armin Hornung, Kai M. Wurm, Maren Bennewitz, C. Stachniss, and Wolfram Burgard. "OctoMap: an efficient probabilistic 3D mapping framework based on octrees." In: *Autonomous Robots* 34 (2013), pp. 189 –206. URL: https://link.springer.com/article/10.1007/s10514-012-9321-0.

[64] Chien-Ming Huang, Maya Cakmak, and Bilge Mutlu. "Adaptive Coordination Strategies for Human-Robot Handovers." In: *Robotics: Science and Systems*. 2015.

[65] Brian Ichter, James Harrison, and Marco Pavone. "Learning Sampling Distributions for Robot Motion Planning." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 7087–7094.

[66] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. "Bc-z: Zero-shot task generalization with robotic imitation learning." In: *Conference on Robot Learning*. PMLR. 2022, pp. 991–1002.

[67] Julius Jankowski, Lara Brudermüller, Nick Hawes, and Sylvain Calinon. "Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior." In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10125–10131. URL: https://arxiv.org/abs/2210.04067.

[68] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions." In: *The International journal of robotics research* 34.7 (2015), pp. 883–921. URL: https://arxiv.org/abs/1306.3532.

[69] Armand Joulin, Laurens van der Maaten, A. Jabri, and Nicolas Vasilache. "Learning Visual Features from Large Weakly Supervised Data." In: *ECCV*. 2016.

[70] Tom Jurgenson and Aviv Tamar. "Harnessing Reinforcement Learning for Neural Motion Planning." In: *Robotics Science and Systems*. 2019.

[71] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. "STOMP: Stochastic trajectory optimization for motion planning." In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 4569–4574. DOI: 10.1109/ICRA.2011.5980280.

[72] Sertaç Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." In: *The International Journal of Robotics Research* 30 (2011), pp. 846 –894.

[73] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. URL: https://ieeexplore.ieee.org/document/508439.

[74] Liyiming Ke, Jingqiang Wang, Tapomayukh Bhattacharjee, Byron Boots, and Siddhartha S. Srinivasa. "Grasping with Chopsticks: Combating Covariate Shift in Model-free Imitation Learning for Fine Manipulation." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 6185–6191.

[75] J Chase Kew, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, and Aleksandra Faust. "Neural collision clearance estimator for batched motion planning." In: *arXiv preprint arXiv:1910.05917* (2019).

[76] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots." In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[77] Alexander Khazatsky et al. "DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset." In: (2024).

[78] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. "Segment anything." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 4015–4026.

[79] Kris M Kitani, Brian D Ziebart, James Andrew Bagnell, and Martial Hebert. "Activity forecasting." In: *European Conference on Computer Vision*. Springer. 2012, pp. 201–214.

[80] Hema S Koppula, Ashesh Jain, and Ashutosh Saxena. "Anticipatory planning for human-robot teams." In: *Experimental robotics*. Springer. 2016, pp. 453–470.

[81] James J. Kuffner and Steven M. LaValle. "RRT-connect: An efficient approach to single-query path planning." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)* 2 (2000), 995–1001 vol.2.

[82] Rahul Kumar, Aditya Mandalika, Sanjiban Choudhury, and Siddhartha S. Srinivasa. "LEGO: Leveraging Experience in Roadmap Generation for Sampling-Based Planning." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 1488–1495.

[83] Tobias Kunz and Mike Stilman. "Time-optimal trajectory generation for path following with bounded acceleration and velocity." In: *Robotics: Science and Systems VIII* (2012), pp. 1–8.

[84] Andras Kupcsik, David Hsu, and Wee Sun Lee. "Learning dynamic robot-to-human object handover from human feedback." In: *Robotics research*. Springer, 2018, pp. 161–176.

[85] SM LaValle. "Planning Algorithms." In: *Cambridge University Press google schola* 2 (2006), pp. 3671–3678. URL: https://lavalle.pl/planning/.

[86] Steven M. LaValle. "Rapidly-exploring random trees : a new tool for path planning." In: *The annual research report* (1998).

[87] Steven M. LaValle. "Planning algorithms." In: 2006.

[88] Steven M LaValle, James J Kuffner, BR Donald, et al. "Rapidly-exploring random trees: Progress and prospects." In: *Algorithmic and computational robotics: new directions* 5 (2001), pp. 293–308. URL: http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf.

[89] Michael Laskey, Jonathan N. Lee, Roy Fox, Anca D. Dragan, and Ken Goldberg. "DART: Noise Injection for Robust Imitation Learning." In: *CoRL*. 2017.

[90] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. "Anytime search in dynamic graphs." In: *Artificial Intelligence* 172.14 (2008), pp. 1613–1643. URL: https://www.sciencedirect.com/science/article/pii/S000437020800060X.

[91] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." In: *ICAPS*. 2005.

[92] Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. "ARA*: Anytime A* with Provable Bounds on Sub-Optimality." In: *NIPS*. 2003.

[93] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization." In: *arXiv preprint arXiv:1711.05101* (2017). URL: https://arxiv.org/abs/1711.05101.

[94] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[95] Guilherme Maeda, Gerhard Neumann, Marco Ewerton, Rudolf Lioutikov, Oliver Kroemer, and Jan Peters. "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks." In: *Autonomous Robots* 41 (2017), pp. 593–612.

[96] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. "Goal set inverse optimal control and iterative re-planning for predicting human reaching motions in shared workspaces." In: *IEEE Transaction on Robotics (TRO)* (2016).

[97] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. "What Matters in Learning from Offline Human Demonstrations for Robot Manipulation." In: *Conference on Robot Learning (CoRL)*. 2021.

[98] Jose R. Medina, Felix Duvallet, Murali Karnam, and Aude Billard. "A human-inspired controller for fluid human-robot handovers." In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)* (2016), pp. 324–331.

[99] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In: *ECCV*. 2020.

[100] Alexander Millane, Helen Oleynikova, Emilie Wirbel, Remo Steiner, Vikram Ramasamy, David Tingdahl, and Roland Siegwart. *nvblox: GPU-Accelerated Incremental Signed Distance Field Mapping*. 2023. arXiv: 2311.00626 [cs.RO]. URL: https://arxiv.org/abs/2311.00626.

[101] Ishan Misra, C. Lawrence Zitnick, Margaret Mitchell, and Ross B. Girshick. "Seeing through the Human Reporting Bias: Visual Classifiers from Noisy Human-Centric Labels." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2930–2939.

[102] Katja Mombaur. "Using optimization to create self-stable human-like running." In: *Robotica* 27.3 (2009), pp. 321–330. URL: https://www.cambridge.org/core/journals/robotica/article/abs/using-optimization-to-create-selfstable-humanlike-running/855871E6530CCB6CA4AB1DADC4CB0DDE.

[103] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. "6-DOF GraspNet: Variational Grasp Generation for Object Manipulation." In: *International Conference on Computer Vision (ICCV)*. 2019.

[104] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. "Continuous-time Gaussian Process Motion Planning via Probabilistic Inference." In: vol. 37. 11. 2018, pp. 1319–1340.

[105] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. "Continuous-time Gaussian process motion

planning via probabilistic inference." In: *The International Journal of Robotics Research* 37 (2018), pp. 1319 –1340.

[106] Mustafa Mukadam, Xinyan Yan, and Byron Boots. "Gaussian Process Motion planning." In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2016-June. Institute of Electrical and Electronics Engineers Inc., June 2016, pp. 9–15. ISBN: 9781467380263. DOI: 10.1109/ICRA.2016.7487091.

[107] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Adam Fishman, and Dieter Fox. "CabiNet: Scaling Neural Collision Detection for Object Rearrangement with Procedural Scene Generation." In: *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2023), pp. 1866–1874. URL: https://arxiv.org/abs/2304.09302.

[108] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. "6-DOF Grasping for Target-driven Object Manipulation in Clutter." In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[109] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. "6-dof grasping for target-driven object manipulation in clutter." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6232–6238. URL: https://arxiv.org/abs/1912.03628.

[110] Kevin Murphy. "A survey of POMDP solution techniques." In: *Environment* 2 (Oct. 2000).

[111] Richard Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking." In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)* (2011).

[112] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning." In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Fran-

cisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, 663–670. ISBN: 1558607072.

[113] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, et al. *Octo: An Open-Source Generalist Robot Policy*. https://octo-models.github.io. 2023. URL: https://octo-models.github.io/.

[114] OpenAI. *ChatGPT-4o*. https://www.openai.com/chatgpt. Accessed: [insert access date here]. 2023.

[115] Andreas Orthey, Constantinos Chamzas, and Lydia E. Kavraki. "Sampling-Based Motion Planning: A Comparative Review." In: *Annual Review of Control, Robotics, and Autonomous Systems* 7.1 (2024), null. DOI: 10.1146/annurev-control-061623-094742. eprint: https://doi.org/10.1146/annurev-control-061623-094742. URL: https://doi.org/10.1146/annurev-control-061623-094742.

[116] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, P. Abbeel, and Jan Peters. "An Algorithmic Perspective on Imitation Learning." In: *Found. Trends Robotics* 7 (2018), pp. 1–179.

[117] Jia Pan and Dinesh Manocha. "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing." In: *The International Journal of Robotics Research* 35.12 (2016), pp. 1477–1496.

[118] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[119] Chris Paxton, Nathan Ratliff, Clemens Eppner, and Dieter Fox. "Representing robot task plans as robust logical-dynamical systems." In: *IEEE Conference on Intelligent Robots and Systems (IROS)* (2019).

[120] Luka Peternel, Wansoo Kim, Jan Babič, and Arash Ajoudani. "Towards ergonomic control of human-robot co-manipulation and handover." In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE. 2017, pp. 55–60.

[121] Dean A. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network." In: *NIPS*. 1988.

[122] C. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 77–85.

[123] C. Qi, L. Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: *NIPS*. 2017.

[124] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." In: *Advances in neural information processing systems* 30 (2017). URL: https://arxiv.org/abs/1706.02413.

[125] A. H. Qureshi, Mayur Joseph Bency, and Michael C. Yip. "Motion Planning Networks." In: *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 2118–2124.

[126] Ahmed H. Qureshi, Arsalan Mousavian, Chris Paxton, Michael C. Yip, and Dieter Fox. "NeRP: Neural Rearrangement Planning for Unknown Objects." In: *ArXiv* abs/2106.01352 (2021).

[127] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision." In: *arXiv:2103.00020*. 2021.

[128] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. "An autonomous dynamic camera method for effective remote teleoperation." In: *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. ACM. 2018, pp. 325–333.

[129] Nathan D. Ratliff, Jan Issac, and Daniel Kappler. "Riemannian Motion Policies." In: *ArXiv* abs/1801.02854 (2018).

[130] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. "Riemannian motion policies." In: *arXiv preprint arXiv:1801.02854* (2018).

[131] Nathan D. Ratliff, Marc Toussaint, and Stefan Schaal. "Understanding the geometry of workspace obstacles in Motion Optimization." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), pp. 4202–4209.

[132] Nathan D. Ratliff, Matthew Zucker, J. Andrew Bagnell, and Siddhartha S. Srinivasa. "CHOMP: Gradient optimization techniques for efficient motion planning." In: *2009 IEEE International Conference on Robotics and Automation* (2009), pp. 489–494.

[133] Nathan Ratliff, Marc Toussaint, and Stefan Schaal. "Understanding the Geometry of Workspace Obstacles in Motion Optimization." In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[134] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. "CHOMP: Gradient optimization techniques for efficient motion planning." In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 489–494. URL: https://ieeexplore.ieee.org/document/5152817.

[135] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. "CHOMP: Gradient optimization techniques for efficient motion planning." In: 2009. DOI: 10.1109/robot.2009.5152817.

[136] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom van de Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. "Learning by Playing Solving Sparse Reward Tasks from Scratch." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, July 2018, pp. 4344–4353. URL: http://proceedings.mlr.press/v80/riedmiller18a.html.

[137] Stephane Ross and Andrew Bagnell. "Efficient reductions for imitation learning." In: *International Conference on Artificial Intelligence and Statistics*. 2010, 661–668.

[138] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *International Conference on Artificial Intelligence and Statistics*. 2010. URL: https://arxiv.org/abs/1011.0686.

[139] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *AISTATS*. 2011.

[140] Stephane Ross, Geoffrey Gordon, and Andrew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning." In: *arXiv preprint arXiv:1011.0686*. 2010.

[141] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision* 115 (2015), pp. 211–252.

[142] Stuart J. Russell. "Learning agents for uncertain environments (extended abstract)." In: *COLT' 98*. 1998.

[143] Kallol Saha, Vishal Reddy Mandadi, Jayaram Reddy, Ajit Srikanth, Aditya Agarwal, Bipasha Sen, Arun Singh, and Madhava Krishna. "EDMP: Ensemble-of-costs-guided Diffusion for Motion Planning." In: *ArXiv* abs/2309.11414 (2023). URL: https://arxiv.org/abs/2309.11414.

[144] Tanner Schmidt, Richard A. Newcombe, and Dieter Fox. "DART: Dense Articulated Real-Time Tracking." In: *Robotics: Science and Systems*. 2014.

[145] John Schulman, Yan Duan, Jonathan Ho, Alex X. Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and P. Abbeel. "Motion planning with sequential convex optimization and convex collision checking." In: *The International Journal of Robotics Research* 33 (2014), pp. 1251 –1270.

[146] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. "Motion planning with sequential convex optimization and convex collision checking." In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270. URL: https://dl.acm.org/doi/10.1177/0278364914528132.

[147] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization." In: June 2013. DOI: 10.15607/RSS.2013.IX.031.

[148] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation." In: *Conference on Robot Learning*. 2022. URL: https://arxiv.org/abs/2209.05451.

[149] David Silver and Joel Veness. "Monte-Carlo planning in large POMDPs." In: *Advances in neural information processing systems*. 2010, pp. 2164–2172.

[150] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. "DESPOT: Online POMDP planning with regularization." In: *Advances in neural information processing systems*. 2013, pp. 1772–1780.

[151] Theodoros Stouraitis, Iordanis Chatzinikolaidis, Michael Gienger, and Sethu Vijayakumar. "Dyadic collaborative Manipulation through Hybrid Trajectory Optimization." In: *Conference on Robot Learning (CoRL)*. 2018, pp. 869–878.

[152] Kyle Strabala, Min Kyung Lee, Anca D. Dragan, Jodi Forlizzi, Siddhartha S. Srinivasa, Maya Cakmak, and Vincenzo Micelli. "Toward seamless human-robot handovers." In: *HRI 2013*. 2013.

[153] Marlin P. Strub and Jonathan D. Gammell. "Adaptively Informed Trees (AIT*): Fast Asymptotically Optimal Path Planning through Adaptive Heuristics." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 3191–3198.

[154] Marlin P. Strub and Jonathan D. Gammell. "Advanced BIT* (ABIT*): Sampling-Based Planning with Advanced Graph-Search Techniques." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 130–136.

[155] Robin A. M. Strudel, Ricardo Garcia Pinel, Justin Carpentier, Jean-Paul Laumond, Ivan Laptev, and Cordelia Schmid. "Learning Obstacle Representations for Neural Motion Planning." In: *CoRL*. 2020.

[156] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library." In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). https://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[157] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. "CuRobo: Parallelized Collision-Free Minimum-Jerk Robot Motion Generation." In: *arXiv preprint arXiv:2310.17274* (2023). URL: https://arxiv.org/abs/2310.17274.

[158] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. "Contact-GraspNet: Efficient 6-DoF Grasp Generation in Cluttered Scenes." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 13438–13444.

[159] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. "Value Iteration Networks." In: *Neural Information Processing Systems*. 2016.

[160] E. Tarados and V. V. Vazirani. *Algorithmic Game Theory, chapter Basic Solution Concepts and Computational Issues*. Cambridge University Press, 2007, 3–28.

[161] Wil B. Thomason, Zachary K. Kingston, and Lydia E. Kavraki. "Motions in Microseconds via Vectorized Sampling-Based Planning." In: *ArXiv* abs/2309.14545 (2023). URL: https://arxiv.org/abs/2309.14545.

[162] Marc Toussaint. "Robot Trajectory Optimization using Approximate Inference." In: *ICML*. 2009, pp. 1049–1056. ISBN: 978-1-60558-516-1.

[163] Marc Toussaint. "Newton methods for k-order Markov Constrained Motion Problems." In: *ArXiv* abs/1407.0414 (2014).

[164] Marc Toussaint. "A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference." In: *Geometric and Numerical Foundations of Movements*. Ed. by Jean-Paul Laumond. Springer, 2017.

[165] Marc Toussaint and Manuel Lopes. "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains." In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 4044–4051.

[166] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. "Training deep networks with synthetic data: Bridging the reality gap by domain randomization." In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 969–977.

[167] Vaibhav V Unhelkar, Ho Chit Siu, and Julie A Shah. "Comparative performance of human and mobile robotic assistants in collaborative fetch-and-deliver tasks." In: *HRI*. IEEE. 2014, pp. 82–89.

[168] Yoji Uno, Mitsuo Kawato, and Ryoji Suzuki. "Formation and control of optimal trajectory in human multijoint arm movement." In: *Biological Cybernetics* 61 (1989), pp. 89–101.

[169] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017). URL: https://arxiv.org/abs/1706.03762.

[170] David Vogt, Simon Stepputtis, Bernhard Jung, and Heni Ben Amor. "One-shot learning of human–robot handovers with triadic interaction meshes." In: *Autonomous Robots* 42.5 (2018), pp. 1053–1065.

[171] Hans S Witsenhausen. "Separation of estimation and control for discrete time systems." In: *Proceedings of the IEEE* 59.11 (1971), pp. 1557–1566.

[172] Patrick Wunsch, Stefan Winkler, and Gerd Hirzinger. "Real-time pose estimation of 3D objects from camera images using neural networks." In: *Proceedings of International Conference on Robotics and Automation* 4 (1997), 3232–3237 vol.4.

[173] Karl Van Wyk et al. "Geometric Fabrics: Generalizing Classical Mechanics to Capture the Physics of Behavior." In: *IEEE Robotics and Automation Letters* 7 (2022), pp. 3202–3209.

[174] Katsu Yamane, Marcel Revfi, and Tamim Asfour. "Synthesizing object receiving motions of humanoid robots with human motion database." In: *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 1629–1636.

[175] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. "Depth anything: Unleashing the power of large-scale unlabeled data." In: *arXiv preprint arXiv:2401.10891* (2024).

[176] Sarah Young, Dhiraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. *Visual Imitation Made Easy*. 2020. arXiv: 2008.04899 [cs.RO].

[177] Xumin Yu, Yongming Rao, Ziyi Wang, Zuyan Liu, Jiwen Lu, and Jie Zhou. "Pointr: Diverse point cloud completion with geometry-aware transformers." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 12498–12507. URL: https://arxiv.org/abs/2108.08839.

[178] Wentao Yuan, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. "M2T2: Multi-Task Masked Transformer for

Object-centric Pick and Place." In: *arXiv preprint arXiv:2311.00926* (2023). URL: https://arxiv.org/abs/2311.00926.

[179] Clark Zhang, Jinwook Huh, and Daniel D. Lee. "Learning Implicit Sampling Distributions for Motion Planning." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 3654–3661.

[180] Zhengyou Zhang. "Parameter estimation techniques: a tutorial with application to conic fitting." In: *Image Vision Comput.* 15 (1997), pp. 59–76.

[181] Tony Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware." In: *ArXiv* abs/2304.13705 (2023). URL: https://arxiv.org/abs/2304.13705.

[182] Tian Zhou and Juan Pablo Wachs. "Early prediction for physical human robot collaboration in the operating room." In: *Autonomous Robots* 42.5 (2018), pp. 977–995.

[183] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. "On the continuity of rotation representations in neural networks." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 5745–5753.

[184] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. "Maximum Entropy Inverse Reinforcement Learning." In: *AAAI*. 2008.

[185] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. "Human Behavior Modeling with Maximum Entropy Inverse Optimal Control." In: *AAAI Spring Symposium on Human Behavior Modeling*. 2009.

[186] Brian D. Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J. Andrew (Drew) Bagnell, Martial Hebert, Anind Dey, and Siddhartha Srinivasa. "Planning-based Prediction for Pedestrians." In: *IROS*. 2009.

[187] Berk Çalli, Arjun Singh, Aaron Walsman, Siddhartha S. Srinivasa, P. Abbeel, and Aaron M. Dollar. "The YCB object and Model set: Towards common benchmarks for manipulation research." In: *2015 International Conference on Advanced Robotics (ICAR)* (2015), pp. 510–517.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of June 7, 2024 (`classicthesis` version 4.2).