Optimistic Active Learning of Task and Action Models for Robotic Manipulation

by

Caris Moses

B.S., Cornell University (2013)
M.S., Northeastern University (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . .
Department of Electrical Engineering and Computer Science
May 13, 2022

Certified by . . . .
Tomás Lozano-Pérez
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by . . . .
Leslie Pack Kaelbling
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Optimistic Active Learning of Task and Action Models for Robotic Manipulation

by

Caris Moses

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Manipulation tasks such as construction and assembly require reasoning over complex object interactions. In order to successfully plan for, execute, and achieve a given task, these interactions must be modeled accurately and capture low-level dynamics. Some examples include modeling how a constrained object (such as a door) moves when grasped, the conditions under which an object will rest stably on another, or the friction constraints that allow an object to be pushed by another object.

Acquiring models of object interactions for planning is a challenge. Existing engineering methods fail to accurately capture how an object's properties such as friction, shape, and mass distribution, effect the success of actions such as pushing and stacking. Therefore, in this work we leverage machine learning as a data-driven approach to acquiring action models, with the hope that one day a robot equipped with a learning strategy and some basic understanding of the world could learn composable action models useful for planning to achieve a myriad of tasks. We see this work as a small step in this direction.

Acquiring accurate models through a data-driven approach requires the robot to conduct a vast amount of information-rich interactions in the world. Collecting data on both real and simulated platforms can be time and cost prohibitive. In this work we take an active learning approach to aid the robot in finding the small subspace of informative actions within the large action space it has available to explore (all motions, grasps, and object interactions). Additionally, we supply the robot with optimistic action models, which are a relaxation of the true dynamics models. These models provide structure by constraining the exploration space in order to improve learning efficiency. Optimistic action models have the additional benefit of being easier to specify than fully accurate action models.

We are generally interested in the scenario in which a robot is given an initial (optimistic) action model, an active learning strategy, and a space of domain-specific problems to generalize over. First, we give a method for learning task models in a bandit problem setting for constrained mechanisms. Our method, Contextual Prior Prediction, enables quick task success at evaluation time through the use of a learned vision-based prior. Then, we give a novel active learning strategy, Sequential Actions, for learning action models for long-horizon manipulation tasks in a block stacking domain. Finally, we give results in a tool use domain for our Sequential Goals method which improves upon Sequential Actions by exploring goal-directed plans at training time.

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Computer Science and Engineering

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering

Acknowledgments

I would like to thank my advisors, Leslie Pack Kaelbling and Tomás Lozano-Pérez. Leslie taught me the importance of a well defined research problem, and Tomás taught me the importance of well thought out robot experiments and the potential challenges associated with them. To my final committee member, Nicholas Roy, thank you for all of your help and guidance during our collaboration with Honda. I would like to thank all three of you for investing in my research career.

To my collaborators, Michael Noseworthy, Sebastian Castro, and Isaiah Brand, I really enjoyed the contributions we made together. Most importantly I want to thank Mike. When we first started working together I was on the verge of dropping out of the PhD program. Having a collaborator who was kind and approachable, easy to talk to and bounce ideas off of, and had more background in machine learning than myself were unquestionably the reasons I was able to make it to the end of this program. I would also like to thank Rachel Holladay and Caelan Garrett for helping me and my collaborators *so much* with getting up and running with PDDLStream and using the Panda robot. I also had the privilege of mentoring some great undergraduates during my time at MIT: Cindy Zhang, Bukunmi Shodipo, and Jack Blazes. I can't wait to see what the future holds for all of you.

Working in the Learning and Intelligent Systems (LIS) group within the EECS department I found some of my closest friends. I would especially like to thank my LIS labmates/friends Gustavo Goretkin and Ariel Anders, and EECS friends Maz Abulnaga (Ω) and Micah Smith (Ω). MIT can be a very intimidating place and you all made it feel warm and welcoming. All of our tea breaks, breakfasts, lunches, Muddy breaks, post it note passing, bad joke telling, yoga classes and everything were exactly what I needed to stay sane. I would also like to thank Teresa Cataldo, Janet Fischer, Alicia Duarte, and Leslie Kolodziejski for their tireless work for LIS and the broader EECS department.

Outside of research I was a part of several other amazing communities including UCEM, the EECS DEI Advisory Group, MSRP, the MLK Luncheon Celebration Planning Committee, and the Black Graduate Student Association (BGSA). In particular I would like to thank Gloria Anglón, Noelle Wakefield, Maria Cervantes Gonzalez, Willie Boag, and Zoe Lemon for your help and dedication to making MIT a better place and celebrating the small black community present here every day.

I started working at the Muddy Charles Pub in 2017 and it immediately became my home away from home. The manager, Mike Grenier, is always so excited when any of the bartenders hit major milestones. Thank you for creating a space that always felt so welcoming, supportive, and warm; all things I desperately needed during the PhD program.

I want to thank all of my friends, especially Angie Lee, Josie Bates, Skye McCaine, John Ok, Kristen Merrigan, Kaitlin Chambers, Chris Luby, Sam Bar, Quincy Pratt, Raspberry Simpson, Grace Jeon, Alex Gaspard-Smith, R'mani Haulcy, Ifueko Igbinedion, and Will Tomlinson for being there for me and cheering me on along the way.

To all of my (some soon to be) family, the McCaine's, the Ariale's, and the Garcia's, thank you for your love and support. To my parent's partners, Miryam Moses and Nate McKoy, you have been amazing additions to our ever growing family and I truly appreciate your love and support. To my grandmother, Barbara Gordon, and my grandparents who weren't able to see this day, Carmen Moses, Claude Gordon, and Septimus Moses, everything I have accomplished is an evolution of you all and your efforts to seek a better life for our family. I hope I have made you proud.

I want to thank my sisters Nina Moses, Cassidy Moses, and Sheri Herbison, for always being there when I needed a break from thinking about research. All of you inspire me every day through your creativity, thoughtfulness, and love. To my dad, Conrad Moses, thank you for always encouraging the scientist in me. I still have memories of you taking me to Women in Engineering fairs as a young child. Everything I have accomplished in the fields of science and engineering is a direct reflection of your investment in and love for me. To my mom, Claudia Moses, thank you for everything. It is a truly special feeling to have someone in your life that you know loves you so unconditionally. You enabled me to succeed by letting me know that if I failed I would be no different in your eyes.

Finally, I want to thank my fiancé Michael Ariale (and Jebbie!). Thank you for being my best friend and confidante through the entire ∼ 6 year journey. I don't think you quite knew what you were getting into when we started dating, but you quickly rose to the challenge of supporting me throughout it all. Thank you for helping me see that life can be silly and making me laugh more than I ever have before. I love the life we have created together and I could not have reached this day without you.

Biography

Caris Moses is a PhD student at MIT working in the Learning and Intelligent Systems Group within the Computer Science and Artificial Intelligence Lab (CSAIL) with Professors Tomás Lozano-Pérez and Leslie Pack Kaelbling. She works at the intersection of robotics and artificial intelligence, and is interested in developing robots that have the ability to reason about low-level physics while trying to accomplish long-horizon manipulation tasks. In her work she explores ways of leveraging active learning to aid in efficient data collection for learning accurate action models. She is passionate about issues of diversity, equity, and inclusion in the spaces she occupies and has been involved with several like-minded organizations including MSRP, the EECS DEI Advisory Group, Black Girls Code, and many others.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic manipulation tasks require reasoning over complex interactions. Humans are able to quickly sense, plan, and act in tasks such as tool use, manipulating constrained mechanisms (such as doors and drawers), and stacking objects in a stable arrangement. We take for granted the fact that we can easily reason over complex continuous spaces to, say, use a hammer or open a door. There are infinitely many ways we could move through open space with a hammer in-hand, and make contact between the hammer and the target object. Additionally, we understand the underlying physical constraints which result in successful hammer use, i.e. applying force in the direction we want the nail to go in to the surface. Planning for and successfully executing a long sequence of actions which involves complex phenomena, such as friction and stability, is a computationally challenging problem in robotic manipulation.

The challenges of long-horizon planning for manipulation have been effectively addressed by task and motion planning (TAMP) methods [95, 99, 33, 49]. These methods work by breaking up the prohibitively large action space into discrete high-level actions. These actions, such as pick and place, require models which describe the preconditions under which executing an action is possible, and effects which will be true once an action is executed. For example, a pick action precondition would be that the robot cannot be holding anything, and an effect would be that the robot is holding something. Assumptions are typically baked into these action models by ignoring the effects of object properties that do not influence the action's success. For example, place actions typically assume that stacking objects such that their geometric centers align will always result in a stable stack, which is accurate when the world consists of blocks with uniform mass distributions. However, it breaks down when the world is more complex and object shapes and mass distributions vary. While these assumptions greatly improve planning efficiency, model misspecification limits a planner's ability to generalize to more complex domains.

While TAMP approaches have worked well in pick-and-place domains, fine-grained manipulation tasks involve complex models which are more difficult to specify by hand. Machine learning (ML) techniques have been used as an effective data-driven solution in these more challenging manipulation scenarios. Methods for learning models for robust grasping [65, 57], peg-in-hole [46, 36], pushing [2, 11], and other highly constrained problems have been successfully developed. Reinforcement and supervised learning techniques have enabled robots to acquire complex models of fine-motor skills. However, these approaches to learning action models typically only work for a single goal-directed task, and do not fit into a larger framework for long-horizon reasoning.

We are interested in expanding upon the capabilities of long-horizon robotic manipulation such that robots can reason about more complex tasks, and understand the underlying physical constraints of a given problem. Integrating ML techniques with TAMP has become an effective approach [103, 92], as ML can be leveraged to learn accurate models of complex dynamics. By augmenting a planning system with the ability to learn from data, we can develop robots capable of overcoming issues associated with model misspecification. We are interested in techniques which enable learned action models to fit easily into high level task and motion planners.

Collecting data on a real or simulated robotic system is both time and cost consuming, and the space of plans that a robot has available to explore is prohibitively large. Randomly executing actions is a very inefficient approach to collecting useful data for learning, particularly for sequential problems where the useful part of the action space is very small relative to the entire action space the robot has to explore.

A visual of this phenomenon for a sequential task is shown in Figure 1-1. To address this we leverage active learning strategies in which the robot is tasked with not only learning some unknown concept, but also with being the experimenter and guiding its own data collection. These strategies allow us to limit the data needed to learn a target concept by guiding experimentation to the useful parts of the action space.

We also aim to use the structure that a task and motion planner provides to improve exploration of long-horizon plans. In this way we do not start learning from scratch, but instead from some simplified but misspecified model. Specifically, we specify *optimistic* models, which are a relaxation of the true unknown dynamics; that is, a model that assumes the robot can successfully reach more states than it actually can. For example, an optimistic pick action might predict that the robot is able to pick up an object from any location, ignoring any underlying object properties which might prevent it from doing so (such as the object's weight). An optimistic pushing model might ignore the conditions required for successful pushing (e.g. friction cone and push direction constraints). Optimistic models are easier to specify than fully accurate transition models as they allow us to ignore the complex conditions under which an action can be successfully executed. A visual of an optimistic relaxation of a grasping model is shown in Figure 1-2. In this work we explore how optimism can be combined with a TAMP formalization to achieve effective and efficient learning of complex robot behaviors.

We are generally interested in understanding what a robot can learn when given an optimistic model, an active learning strategy, and an evaluation task which is different from the training tasks in some way. First, in Chapter 2 we give problem formulations for the concrete problems addressed in this thesis for learning both task and action models. In Chapter 3 we discuss related work. Chapter 4 gives the background of specific related work which we leverage in our work. In Chapter 5 we give a method, Contextual Prior Prediction, for learning task models which enable a robot to generalize to novel tasks at evaluation time. This work uses a bandit framework where goals can always be achieved with a single action. As such, the learned models do not fit into a long-horizon TAMP system. However, this method

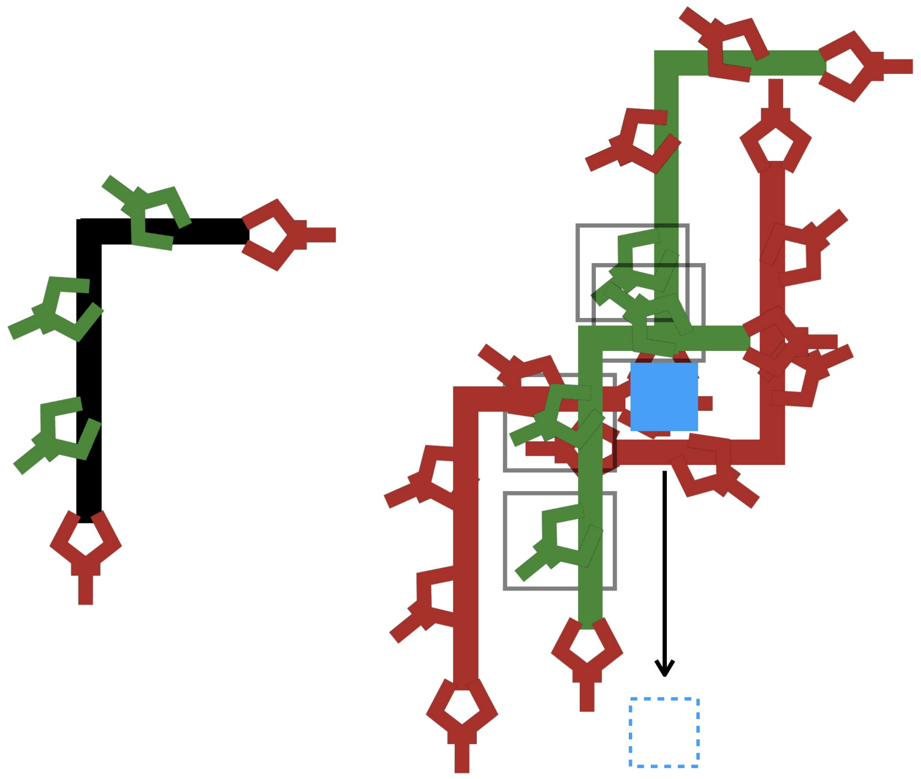


Figure 1-1: **Size of successful actions subspace for a single action versus sequential actions.** In this example the robot is tasked with moving the blue block to the blue dotted region by using the black hook-shaped tool and moving downward in a straight line. On the left we show successful (green gripper) and unsuccessful (red gripper) attempts at grasping the tool from its initial position. On the right we show contact configurations between the tool and the block which would successfully (green tool) or unsuccessfully (red tool) move the blue block to the goal position under friction cone constraints. When the robot attempts to move the block with the tool, some of the grasps which previously worked for picking up the tool no longer work. This is either due to the contact configuration between the tool and block not being successful, or the gripper goal pose being kinematically out of reach. When the task was simply grasping the tool (shown on the left), the successful actions were a larger proportion of all actions the robot attempted. However, when we attempt the longer problem of tool use (shown on the right), the number of actions which successfully accomplish the task shrink in proportion to the full action space the robot has to explore. The few remaining successful grasps are shown in gray squares.

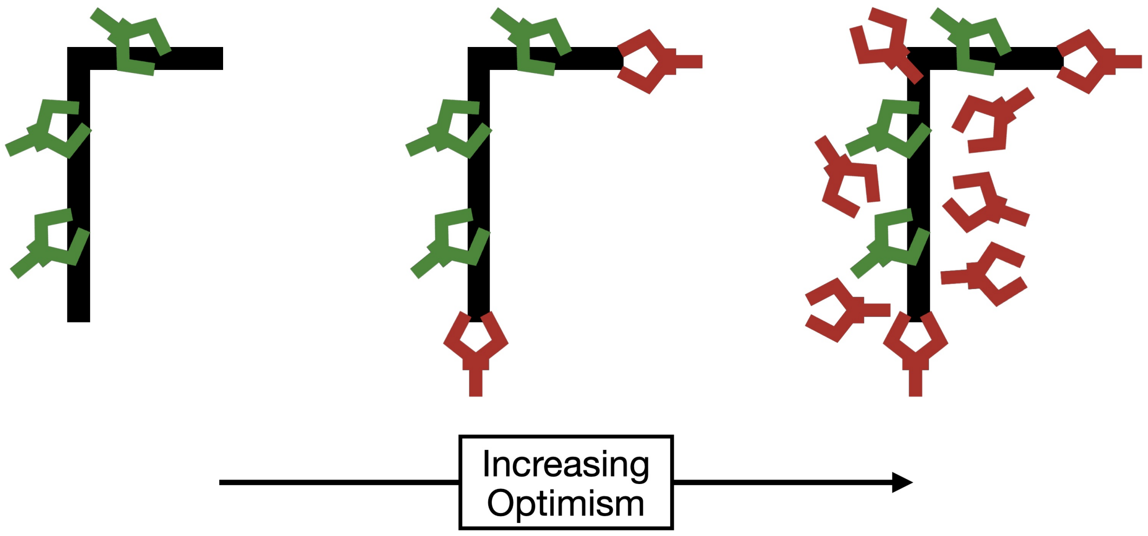


Figure 1-2: **Increasingly optimistic grasping models.** This shows different optimistic grasping models for a robot attempting to grasp and pick up the black hook-shaped tool. It attempts all shown grasps *optimistically*, believing that they will work and that the tool will remain in the gripper once it is picked up. The grasps shown in green result in successful grasps while those in red are unsuccessful attempts. The grasping model on the left corresponds to the underlying true grasping model and all attempted grasps work. The middle model shows a relaxation of this in which we assume that simply making contact with the tool will result in successful grasps, but learn that this assumption does not hold for two of the attempted grasps. Finally, on the right we show a relaxation on the contact assumption in which the robot believes all visualized grasps will work. Here the successful grasps are an even smaller set of grasps within all attempted grasps.

still provides insight into successful active learning from an optimistic model. We then give methods which address both active learning of action models and long-horizon planning in Chapters 6 and 7. In Chapter 6 we give an active learning strategy, Sequential Actions, for learning action models in sequential domains. In Chapter 7 we introduce Sequential Goals, which improves upon Sequential Actions by leveraging goal-directed exploration at training time. We evaluate the methods discussed in Chapters 5, 6, and 7 in a constrained mechanisms, block stacking, and tool use domain, respectively. Finally, we conclude and summarize our findings in Chapter 8.

Chapter 2

Problem Formulation

In this chapter we discuss the problem formulations addressed by this work. At a high level we address the problem in which a robot is given a misspecified *optimistic* action model and must learn an accurate model which is useful given an evaluation task. Our approaches can be broken down into two phases: a learning and evaluation-phase, as shown in Figure 2-1. During the learning-phase the agent uses the optimistic model within an active learning strategy to collect data and learn an accurate model. Then, during the evaluation-phase, the robot uses this accurate model to plan for and achieve a goal, given the initial state. In our work the learned accurate model is either a task feasibility model (TFM) or an action feasibility model (AFM). We give problem formulations for both in Sections 2.2 and 2.3 respectively. First, we give the definition of an optimistic model in Section 2.1.

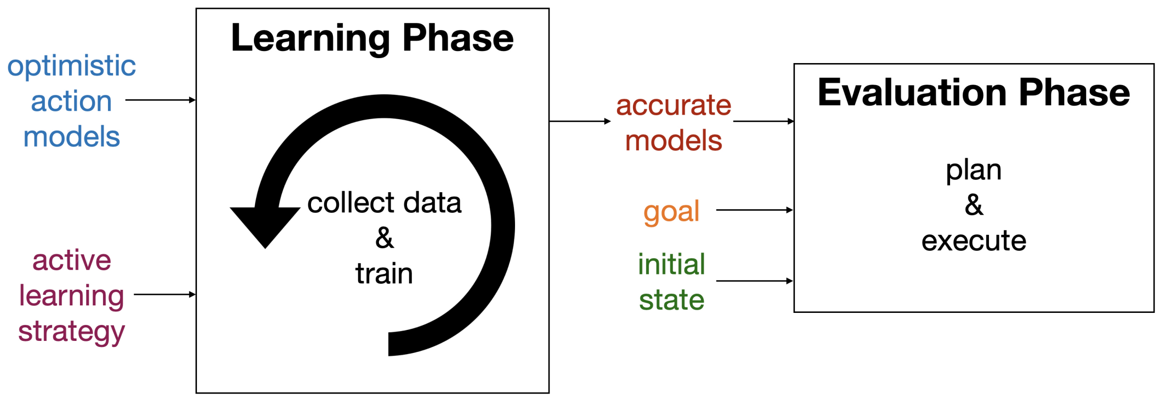


Figure 2-1: **High level approach.**

2.1 Optimistic Model

A robot operates in an observable state space $\mathcal{S}$ and can take actions in action space $\mathcal{A}$. It operates under an *unknown* true transition function $f_T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ and is given an *optimistic* transition function, $f_O : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$.

The set of states reachable under the optimistic transition function from state s_{i-1} is

$$\mathcal{S}_O(s_{i-1}) = \{s_i \mid \exists a \text{ s.t. } s_i = f_O(s_{i-1}, a)\},$$

and the set of states reachable under the true transition function from state s_{i-1} is

$$\mathcal{S}_T(s_{i-1}) = \{s_i \mid \exists a \text{ s.t. } s_i = f_T(s_{i-1}, a)\}.$$

When we say that $\mathcal{S}_O$ is an optimistic version of $\mathcal{S}_T$, we mean that these functions have the following property

$$\mathcal{S}_T(s) \subseteq \mathcal{S}_O(s).$$

Intuitively, f_O is a relaxation of f_T, and believes that the robot can transition into more states than it actually can within a single action step. It is important to note that planning with f_O may allow plans to be constructed that will not actually be executable under the true transition dynamics.

2.2 Task Feasibility Model

First, we examine a multi-armed bandit setting in which the environment resets to the initial state after each action is attempted. At evaluation time, given a novel state, s, the robot must find the action, a^*, which maximizes the unknown task reward function, $R_{task} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$,

$$a^* = \operatorname*{argmax}_{a \in \mathcal{A}} R_{task}(s, a).$$

We take a data-driven approach to this problem and propose to learn an approximation of the unknown reward function which we refer to as a *task feasibility model* (TFM).

2.2.1 Learning Problem

We use the optimistic model to explore and collect data to train an approximation of the unknown reward function or TFM, $\hat{R}_{task} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ [74]. We learn $\hat{R}_{task}$ by collecting a dataset of (s, a, r) tuples where r is the reward received after taking action a from state s. We give a method for learning $\hat{R}_{task}$ and using it in the evaluation-phase in Chapter 5.

2.3 Action Feasibility Model

We then look into sequential decision-making problems where the robot is given a task which requires a sequence of actions. We present two different evaluation-phase problems. In the first, the robot is given the task in the form of a reward function, $R_{task} : \mathcal{S} \rightarrow \mathbb{R}$, and must find a plan in which the last state maximizes reward,

$$a^*_{1:N} = \operatorname*{argmax}_{a_{1:N} \in P} R_{task}(s_N).$$

In the second, the robot is given a goal state, $s_{goal} \in \mathcal{S}$, and must find a plan which brings the robot to the goal state

$$a^*_{1:N} = \operatorname*{argmax}_{a_{1:N} \in P} \mathbb{1}_{\{s=s_{goal}\}}(s_N).$$

We note that in many of our tasks the length of plans in the search space P varies, meaning the value N varies as well. In our work in Chapter 6, we constrain the planner to only generate plans of a certain length N, and in Chapter 7 we use a skeleton-based planner where N is determined by the length of the given skeleton.

Maximizing these objectives with a plan space P generated using the optimistic model would potentially result in an infeasible plan and unsuccessful plan execution. This is due to the fact that the optimistic model is not accurate. To successfully plan to accomplish a given task, we need the help of an additional model, a probabilistic classifier, which indicates the probability that the optimistic model's prediction is correct, or feasible. We refer to this classifier as an *action feasibility model* (AFM),

$$f_A : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1],$$

which predicts the probability that taking an action from a state is accurately modeled under the optimistic transition function. We can evaluate the feasibility of an optimistic plan, $a_{1:N}$, from a given initial state, s_0, as

$$\text{PF}(s_0, a_{1:N}) = \prod_{i=1}^{N} f_A(s_{i-1}, a_i) \tag{2.1}$$

where $s_i = f_O(s_{i-1}, a_i)$.

Now at evaluation time we can use the feasibility model to find plans that maximize the combination of task reward and plan feasibility

$$a_{1:N} = \underset{a_{1:N} \in P}{\text{argmax}}\, \text{PF}(s_0, a_{1:N}) R_{task}(s_N).$$

And in the case where the robot is given a goal state we can do the same,

$$a_{1:N} = \underset{a_{1:N} \in P}{\text{argmax}}\, \text{PF}(s_0, a_{1:N}) \mathbb{1}_{\{s = s_{goal}\}}(s_N).$$

2.3.1 Learning Problem

We propose to learn the parameters of the action feasibility model by performing supervised learning. During data collection the robot takes an action a from a state s_{i-1}, transitions to state s_i^T under the true unknown dynamics, and receives feasibility label

$$\phi = \begin{cases} 1 & \textsc{close}(s_i^T, s_i^O) \\ 0 & \text{otherwise,} \end{cases}$$

where s_i^O is the state which the optimistic model expected to transition to. For continuous state spaces such as object poses, $\textsc{close}(s^T, s^O)$ evaluates to True if the states are within some distance ϵ of each other. For logical states, $\textsc{close}(s^T, s^O)$ evaluates to True if the logical states match, for example whether or not a robot is holding a given object.

The dataset of (s, a, ϕ) samples are used to train the approximated action feasibility model,

$$\hat{f}_A : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1].$$

The predictions are probabilistic to account for our epistemic uncertainty, or the uncertainty we have over our predictions. We will discuss our methods, Sequential Actions and Sequential Goals, which address this learning problem in Chapters 6 and 7.

Chapter 3

Related Work

Our work touches on many topics in the areas of learning and planning for robotic manipulation. There have been many recent advances in applying machine learning to robotics problems. However, they typically address low-level manipulation tasks [25, 64, 2]. In this work we aim to enable a robot to learn skills which can be integrated into a planning framework for long-horizon problems. We also aim to do this in a data efficient manner. In Section 3.1 we discuss related works in the space of active learning where others are also concerned with learning efficiency. In Section 3.2 we discuss the problem of learning task-specific models. Section 3.3 addresses learning models for planning, works most closely related to ours. Finally, in Section 3.4 we discuss the role of optimism in machine learning applications.

3.1 Active Learning

Active learning [70] is a well-established learning paradigm that aims to minimize the number of samples needed to learn an unknown function. Active learning methods are leveraged in robotic manipulation learning problems [19, 103] in order to limit the amount of data collection required on robotic platforms, a process which often requires vast amounts of time and human supervision.

Bayesian Active Learning by Disagreement (BALD) [45] is a method which takes a Bayesian approach to active learning by not just learning the unknown function, but additionally modeling uncertainty over the parameters of the unknown function. This method selects training samples in areas of the model input space where there is high disagreement over the model parameter values. We leverage this strategy in our work, and give a brief overview of this method in Section 4.2. Gal et al. [30] extends BALD to high-dimensional domains by using deep neural networks and MC-dropout [29] to approximate the distribution over the unknown model parameters. In our work on learning action feasibility models, discussed in Chapters 6 and 7, we follow the approach of Beluch et al. [12] and use an ensemble of deep networks to approximate this distribution.

Another strategy for active learning which we leverage in our work is Gaussian Process Upper Confidence Bound (GP-UCB), where the objective is to maximize an unknown function. The function is modeled using a Gaussian Process (GP), and an acquisition function is used to select training data by both exploiting areas of the input space which have high mean value, and exploring parts of the input space which have high variance. We give a brief overview of this method in Section 4.1.

Ideas from active learning have also been used in model-based reinforcement learning (RL) tasks [85, 90]. Baranes et al. [7] address active exploration via a *competence* metric which is used to scores goals. The robot then executes a plan to achieve the highest scoring goal with the purpose of learning inverse action models. The competence metric is high in parts of the goal space where the robot's ability to achieve goals is changing. Similar to the BALD acquisition strategy leveraged in our work [45], the *disagreement* concept can be applied to a model-based RL setting. Pathak et al. [85] represent a forward dynamics model with an ensemble of networks and use the variance of the ensemble output as intrinsic motivation to guide data collection. In our work instead of predicting continuous states we predict action feasibility, and use it in combination with an optimistic model to search for feasible plans.

3.2 Learning Task Models

Our work on task feasibility model learning, given in Chapter 5, addresses problems that are often addressed by RL methods. We use pixels as input to the learned model which is common in RL applications [85, 25, 64]. Where we seek to learn a model which can generalize to many initial states, or images, with shared structure, RL methods typically aim to learn a policy for a single initial state and goal. In this setting, ideally a robot could be trained to do any task with basic visual sensing and the correct learning framework, eliminating the need to manually engineer perception and control systems. Finn et al. [25, 64] train a network to perform various manipulation tasks from pixel-space input. Their work focuses on learning to perform a single task well, whereas we focus on manipulating multiple objects that share structure.

Similar to Agrawal et al. [2], we use a model-based policy in our work on learning task models. They focus on learning forward and inverse models for poking objects where the robot is given low-level actions parameterized by a direction and distance to move the end effector. We assume we have a structured parameterized policy space based on kinematic models and focus on learning which parameterization is correct for a given object. Other model-based work attempts to learn a dynamics model to be used in a controller [62, 27, 76].

Recent work has started to explore the idea of learning an embedded space of policies to handle the more general case of policy learning for a variety of tasks in a single environment. Lynch et al. [69] develop a method for learning from teleoperated "play" data in which a human controls a simulated robot to generate data to train an embedded policy space. Hausman et al. [42] use a similar objective, but the data is generated via off-policy reinforcement learning. In both works the agent is constrained to perform well in only a single environment.

3.3 Learning for Task and Motion Planning

Learning accurate models for long-horizon planning is an increasingly active area of research. Models for planning are typically represented by either fully symbolic or hybrid (parameterized by both discrete and continuous) parameters. Methods for learning symbolic action models which are either deterministic [17] or noisy [83] have been explored via various learning strategies such as random goal sampling [17]. The work of Wang et al. [103] is most similar to our action feasibility model learning work in that they assume a TAMP architecture with hybrid actions, and learn constraints on continuous parameters of hybrid action models. A similar approach is taken in [50], a precursor to [103]. Silver et al. [92] learn full probabilistic hybrid action models by clustering observed transitions.

A closely related body of work to learning accurate action models for planning, is learning models to aid in planning *given* accurate action models [19, 108, 55, 56, 54, 53]. These methods learn mappings from action types and goals to successful action parameterizations. In Xu et al.'s [108] work this is referred to as an actions' affordance.

In all methods referred to so far, as well as in our work, the training and evaluation problems are drawn from the same distribution. However methods vary in how this distribution is represented. One could vary the objects, and initial and goal states between training and evaluation [103]. Another evaluation approach is to maintain the same objects but vary the initial and goal states between training and evaluation [17, 92]. Finally, as is typical in RL settings, one could use the same objects, initial and goals states during training and evaluation [19, 108], although Curtis et al. [19] also gives results on transferring the learned models to different problems.

In our work we explore learning from *optimistic* action models, but learning from models which are inaccurate in other ways has also been explored. Lagrassa et al. [59] use model-free RL to augment a given inaccurate planning domain when an unexpected transition occurs. Other methods have been used to learn residual controllers on top of inaccurate controllers, where the inaccuracies stem from either simulation approximations to the real world [3], or poorly learned RL-based controllers [91].

3.4 The Role of Optimism in Learning

Optimism in the face of uncertainty has been widely studied for multi-armed bandit problems [94, 6, 20, 75, 5] as well as both model-free and model-based RL problems [98, 79, 82, 75, 5]. In these problem settings the objective is to maximize cumulative reward (minimize cumulative regret) or final reward (final regret) given a computation budget. We have the same objective in our work on learning task feasibility models. In our action feasibility model learning work the objective is to learn an accurate model which can be used in a planner to achieve a variety of goals. Even though the objectives are different, in both cases exploration is necessary to achieve the desired outcome, and thus previous work in the area of optimism is relevant and vitally important.

In the case of multi-armed bandits where the objective is to minimize cumulative or final regret, confidence bounds have been found to be very effective at handling the exploration-exploitation trade off. Here the confidence bounds act as an optimistic estimate of a selected action's value [6, 20].

In RL, optimism can be used as a form of exploration by either initially making all actions [98, 79] or states [82] look optimistically rewarding. In the work of Osband et al. [79] they theoretically prove that in most cases, exploring by optimistically sampling from a model posterior is more effective than artificially initializing a model to be maximally rewarding everywhere, and refer to optimistically sampling from a posterior as *stochastic optimism.* This insight is present in our work in that when we score potential actions, we do so by sampling from our current feasibility model posteriors. We do not use the optimistic model to make the action space look artificially rewarding everywhere for exploration, but rather to represent the space of actions the robot has available to explore.

Auer et al. [5] and Munos et al. [75] develop algorithms for addressing both bandit and RL problems with the use of optimistic exploration, and give performance bounds in terms of the complexity of the domain [75] and the adversarial “shift” used during exploration [5].

Chapter 4

Background

4.1 Gaussian Process Upper Confidence Bounds

The Gaussian Process (GP) provides a nonparametric method for modeling an unknown function as a distribution over functions

$$f(x) \sim \text{GP}(\mu(x), k(x, x'))$$

where $\mu(x)$ is the mean function, and $k(x, x')$ is a kernel function which models the covariance between pairs of function values [86]. GP regression allows us to calculate the posterior distribution given a dataset of input output pairs $\mathcal{D} = (\mathbf{x}, \mathbf{y})$.

$$\begin{aligned}
&f(x) \sim \text{GP}(\mu_{\mathcal{D}}(x), k_{\mathcal{D}}(x, x)) \\
&\mu_{\mathcal{D}}(x) = \mu(x) + k(x, \mathbf{x})\mathbf{K}^{-1}(\mathbf{y} - \mu(\mathbf{x})) \\
&k_{\mathcal{D}}(x, x') = k(x, x') - k(x, \mathbf{x})\mathbf{K}^{-1}k(\mathbf{x}, x') \\
&\mathbf{K} = k(\mathbf{x}, \mathbf{x})
\end{aligned} \tag{4.1}$$

Gaussian Process Upper Confidence Bound (GP-UCB) is an active learning strategy which aims to maximize an unknown function. It does this by selecting points which both exploit areas of the input space which have high mean value, and exploring areas with high variance. This allows for the agent to maximize the unknown function such that there is low uncertainty everywhere, and higher resolution in the parts of the input space with high value. We leverage this method when learning a task feasibility model where the ultimate goal is to maximize reward, thus requiring a good understanding of these parts of the input space. Given the posterior distribution shown in Equation 4.1, the GP-UCB acquisition function [94] generates an input sample x^* using the following objective

$$x^* = \operatorname*{argmax}_x \mu_{\mathcal{D}}(x) + \beta^{1/2} k_{\mathcal{D}}(x, x) \tag{4.2}$$

where β is a parameter which determines how much to factor in uncertainty when selecting the next input.

4.2 Bayesian Active Learning by Disagreement

Bayesian Active Learning by Disagreement (BALD) [45] gives a strategy for learning an unknown function parameterized by Θ. In this framework we take a Bayesian approach and maintain a distribution over the unknown model parameters $\Pr(\Theta \mid \mathcal{D})$. The goal is to select the dataset, $\mathcal{D} = (\mathbf{x}, \mathbf{y})$, that maximally reduces the entropy of $\Pr(\Theta \mid \mathcal{D})$,

$$\operatorname*{argmin}_{\mathcal{D}} \mathcal{H}(\Theta \mid \mathcal{D}).$$

The general problem of designing a sequence of experiments to minimize entropy — or equivalently, maximize information gain — is an NP-hard sequential decision-making problem [43]. Fortunately, in submodular problems, a myopic approach that considers only the next experiment to conduct can be shown to be a good approximation to the optimal experimentation strategy [26]. We can then greedily select a single datapoint, (x, y), at a time

$$\operatorname*{argmax}_x \mathcal{H}(\Theta \mid \mathcal{D}) - \mathcal{H}_{y \sim \Pr(\cdot \mid x, \mathcal{D})}(\Theta \mid y, x, \mathcal{D}). \tag{4.3}$$

However computing entropy in the parameter space Θ is generally intractable, especially in our case where Θ are the parameters of an ensemble of neural networks as will be discussed in Section 6.1. Houlsby et al. [45] note that Equation 4.3 is equal to the mutual information between the unknown output and model parameters, $\mathbb{I}(\Theta; y \mid x, \mathcal{D})$. Therefore they reformulate the objective in the low-dimensional output space, yielding the final objective

$$\textsc{bald}(x) = \mathcal{H}(y \mid \mathcal{D}, x) - \mathbb{E}_{\Theta \sim \Pr(\cdot \mid \mathcal{D})} \left[\mathcal{H}(y \mid x; \Theta) \right]. \tag{4.4}$$

Initially, the distributions over both the model parameters, $\Pr(\Theta \mid \mathcal{D})$, and outputs, $\Pr(y \mid x, \Theta)$, represent the uncertainty we have with respect to our predictions. This uncertainty is referred to as *epistemic* uncertainty. *Aleatoric* uncertainty is the the uncertainty in an underlying stochastic process. The goal of selecting inputs via the BALD objective, is to eliminate *epistemic* uncertainty, such that in the end any remaining uncertainty is captured by $\Pr(y \mid x, \Theta)$ and only represents *aleatoric* uncertainty.

Finding the input, x, which maximizes Equation 4.4 invites an appealing interpretation: maximizing the first term encourages selecting an x where our overall uncertainty of the output value is high, and minimizing the second term encourages selecting an x for which there is high confidence in varying parts of the model parameter space. For there to be both high overall uncertainty and high individual model confidence, the models must be in *disagreement*. Intuitively, if our distribution $\Pr(\Theta \mid \mathcal{D})$ has areas of high confidence in different parts of the output space, then observing a y value in one of these areas is likely to prove some of those predicted outcomes incorrect. If we think of the overall uncertainty as a combination of *epistemic* and *aleatoric* uncertainty, then this objective seeks an experiment with high overall uncertainty and low *aleatoric uncertainty*, which therefore has high *epistemic* uncertainty.

4.3 Task and Motion Planning

Task and motion planning methods [31, 32, 95, 99, 21, 61, 60, 101, 33, 24, 96, 31, 15, 48, 49, 67, 106] enable robots to plan for long-horizon manipulation problems which involve actions parameterized by both discrete and continuous action parameters (hybrid actions). Many of these methods represent the discrete and semantic aspects of the planning problem using the Planning Domain Definition Language (PDDL) [31, 32, 95, 99, 21, 61, 33, 96, 31, 15, 106].

4.3.1 PDDLStream

PDDLStream [33] gives a method for extending PDDL to the continuous aspect of robotic manipulation through the use of planning streams. The details of how PDDLStream finds a concrete plan for a given planning problem are discussed in [33]. Here we simply describe how to define a planning problem. The following notation borrows heavily from Garrett et al. [33].

A predicate p is a boolean function, and an atomic fact $p(\mathbf{x})$ is a predicate evaluated on object tuple $\mathbf{x}$ which evaluates to True. A literal is a fact or a negated fact. A state, s, is a set of literals. We operate under the closed world assumption, meaning that facts not explicitly in the state are False. An action is specified by its parameters $\mathbf{X}_{A_i}$, a set of literal preconditions $pre_{A_i}(\mathbf{X}_{A_i})$, and a set of literal effects $eff_{A_i}(\mathbf{X}_{A_i})$. During planning an action is applicable in state s if $pre^{+}_{A_i}(\mathbf{x}) \subseteq s$ and $pre^{-}_{A_i}(\mathbf{x}) \cap s = \varnothing$, where the $+$ and $-$ superscripts designate the positive and negative literals respectively. In our work we define a planning problem with an initial state, a goal state, and a set of actions for a given domain. Stream functions are functions which take in action parameters and return potential values for continuous action parameters. They are also part of the planning problem, but we omit them in this thesis for simplicity.

In Sections 6.2.2 and 7.2 we give the states and actions used in a block stacking and tool use domain, respectively. Note that in the problem formulations given in Chapter 2, the optimistic transition function, f_O, is given in the functional representation, $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, however in our work in Chapter 7 the optimistic model is represented by PDDLStream actions.

Chapter 5

Task Feasibility Model Learning

As humans, we frequently encounter new objects and are able to successfully articulate them with little to no experience on those particular object instances.[1] Consider entering a new kitchen. You can quickly open the cupboards, turn on the lights, and operate the stove, even though you have never used these specific objects before. This behavior is enabled by rich visual cues such as the presence of a handle, or the location of hinges on a door (see Figure 5-1). These features are useful for inferring the function of a new mechanism and for estimating the motion that it can undergo.

Our goal is to enable a robot to efficiently interact with novel articulated objects without human guidance by learning from previous visuo-motor experience with related objects. Previous work has shown how robots can infer the kinematic models of new mechanisms given a single demonstration of the mechanism being actuated [97]. However, demonstrations are often expensive as they require a human teacher every time the robot needs to interact with a new object. Other methods provide exploration strategies which enable a robot to estimate the kinematic properties of mechanisms [8], [80]. While these methods perform well, they do not transfer any experience from similar mechanisms when interacting with novel mechanisms. We desire a solution where the robot uses past experience to experiment efficiently with

[1]The material in this chapter is based on:
Caris Moses*, Michael Noseworthy*, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Visual Prediction of Priors for Articulated Object Interaction. IEEE International Conference on Robotics and Automation (ICRA), 2020. [74]

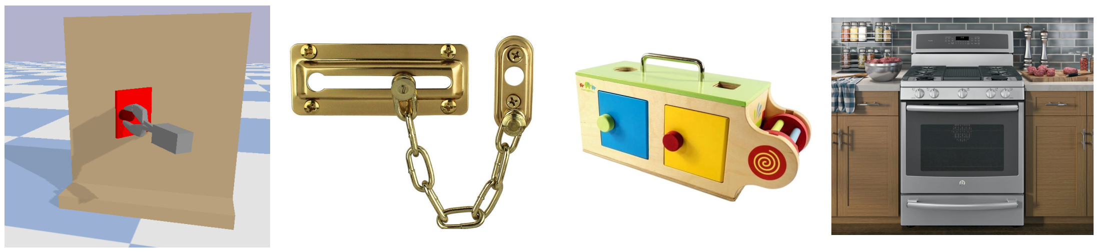


Figure 5-1: **Objects with rich visual features.** Contextual Prior Prediction (CPP) is motivated by the fact that most objects have rich visual features which indicate their motion. In our simulated door domain (left), the position of the handle and width of the door indicate the direction which it opens and with what radius. Other objects, such as latches (center left), toys (center right), or ovens (right) also have visible indicators such as tracks, hinges, and knobs.

new mechanism instances to learn how to actuate them from a very small number of self-selected actuations.

Due to constraints present in articulated objects, very few of the possible motion commands the robot can generate are likely to cause the mechanism to move. Without models of the object, the robot must propose its own goals or sequences of actions that can quickly generate motion that exhibits the correct kinematic structure of the mechanism. In order to enable efficient exploration, we propose our method, *Contextual Prior Prediction* (CPP), which uses the visual appearance of a mechanism to provide a prior that indicates which actions are likely to be successful in actuating the mechanism. The mapping from visual appearance to an actuation prior is learned from previous interactions with mechanisms from the same class. This enables the robot to only try actions that it believes are likely to succeed based on the new object's appearance. We also found initializing the space the robot has to explore to an optimistic action space to be very useful for data efficiency. Specifically, we use task knowledge to limit the exploration space to tasks of the same type but unknown parameters. For example, when exploring how to actuate a sliding joint, the robot attempts to actuate it as if it were a sliding joint with various angles and lengths.

Referring back to our high level approach shown in Figure 2-1, we have a learning-phase and an evaluation-phase. During the learning-phase the robot learns a general mapping from the appearance of a mechanism and a proposed action to a reward which measures how far the mechanism moves, i.e. a *task feasibility model* (TFM) or reward function. Then during the evaluation-phase, given a novel object and starting with a prediction based on its visual appearance and previous interactions, the robot must actively select a sequence of motion commands that will cause the mechanism to move. As the robot gets more experience in the learning-phase with different mechanisms, it is able to “understand” a new mechanism with fewer and fewer trials in the evaluation-phase.

We formulate the overall problem as a Contextual Multi-Armed Bandit (C-MAB) in which the robot continuously interacts with a sequence of mechanisms (contexts) with shared structure. For the learning-phase, we represent the TFM using a neural network, which maps visual appearances and possible actions to value, and train it using conventional supervised learning methods. Then, for the evaluation-phase, given a novel mechanism to interact with, we use the TFM to predict the expected value of each possible action in a continuous action space. We treat this function as the prior mean of a Gaussian Process (GP), and use the Gaussian Process Upper Confidence Bound (GP-UCB, details in Section 4.1) strategy [94] to handle the exploration-exploitation trade off when finding the optimal action.

In our method, GP-UCB is also used in the learning-phase. We explore the effectiveness of GP-UCB and random sampling as exploration strategies during training. At evaluation time, we compare the overall effectiveness of the system to one that applies GP-UCB to each new mechanism starting from a generic prior, as well as to a baseline random search. The techniques are generic and could apply to a variety of mechanisms. Our experimental comparisons are done in a simulated domain containing prismatic and revolute joints with different visual appearances and kinematic parameters. The learned reward functions do not fit into a long-horizon planning system, but the methods investigated here give good insight into how we can successfully explore an optimistic action space using active learning.

The contribution of this work is to demonstrate how a period of exploration with mechanisms with different visual appearances can lead to the ability to actuate never-before-seen mechanism instances with very few (sometimes just one) trials. In addition, the approach of learning to map appearance to the prior of a GP, rather than mapping appearance directly to motor commands, means that the overall system is significantly more robust, and can recover from inaccurate predictions that arise when there is little training data.

5.1 Method

We propose a method to address the problem of task feasibility model learning, as laid out in Section 2.2. The problem is modeled as a C-MAB, in which the robot is given a novel initial state represented by an image, and must maximize reward. We address this problem by first collecting data in order to learn a task-specific model, $\hat{R}_{task}$. Then at evaluation time, given a novel initial state we use $\hat{R}_{task}$ within an active learning strategy to maximize reward. An image of the learning and evaluation-phase is given in Figure 5-2

5.1.1 Training Phase

During the learning-phase for a given task, the robot is presented with a sequence of initial states, $s \in \mathcal{S}$, and uses active learning within the space of the optimistic model, f_O, to select an action, $a \in \mathcal{A}$. It gets M chances to interact with the environment, receiving a reward, $r \in \mathbb{R}$, and resetting to the initial state after each interaction. The generated dataset of (s, a, r) tuples are used to train the TFM, $\hat{R}_{task}$.

5.1.2 Evaluation Phase

At evaluation time the robot is presented with novel initial states and must maximize reward. A naive strategy would simply find the action which maximizes the reward of the learned model given an new state s

$$a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \hat{R}_{task}(s, a).$$

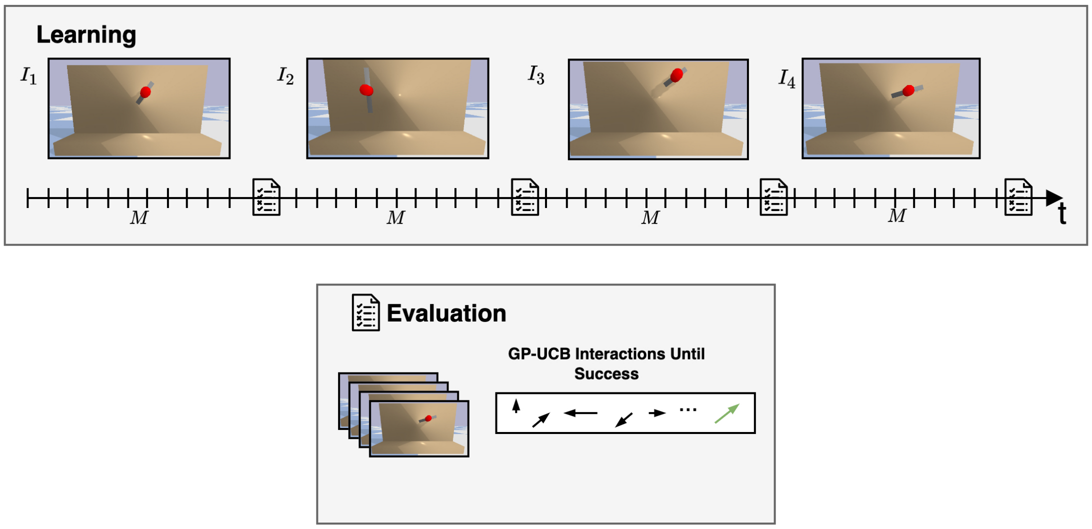
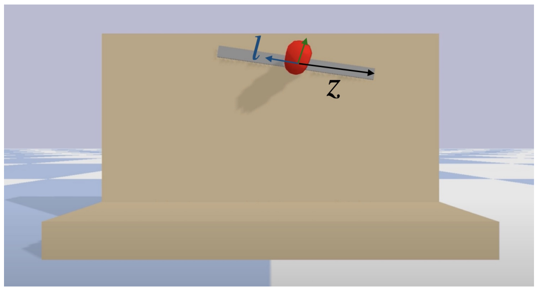


Figure 5-2: **Learning and evaluation-phase for Contextual Prior Prediction.** During the learning-phase, the robot is sequentially presented with mechanisms. The robot can interact with each mechanism for M steps (timeline ticks) before a new one appears. We evaluate the robot's performance on a separate evaluation set of novel mechanisms (bottom). For each evaluation mechanism, the robot takes actions until it has generated an optimal interaction (generated the most possible motion).

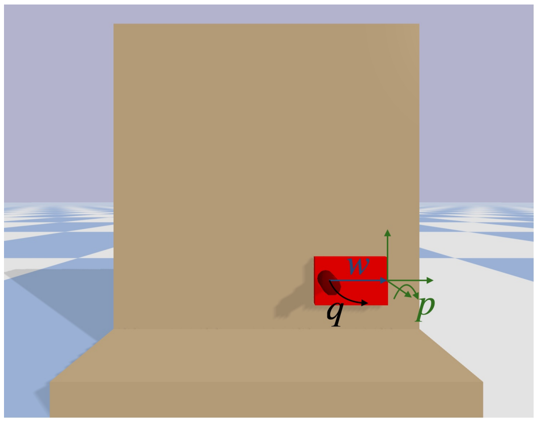
However, when we have a small amount of training data, this prediction may not be very accurate. We would like to extend the model such that when the robot takes actions and fails, the rewards from failed attempts inform future interactions as the robot searches for the optimal action. We do so by modeling each new initial state with its own GP (Equation 4.1) where the mean function is the learned task model, and use GP-UCB (Equation 4.2) to collect data. This helps to provide guidance before any new data is collected. We refer to this as *Contextual Prior Prediction* (CPP) [74].

Specifically, given a new state, $s \in \mathcal{S}$, we use a GP (Equation 4.1) to model a state-specific reward function, where the input space is an action, $x \in \mathcal{A}$, and the output is the reward, $y \in \mathbb{R}$. We then use GP-UCB (Equation 4.2) to select datapoints with the initial mean function set to the learned model, $\mu(x) = \hat{R}_{task}(s, x)$.

Under this criterion, samples initially come from parts of the space where the learned reward function predicts high reward. Then as the true state/context-specific reward function is learned, we select actions with much more accurate knowledge of



(a) Slider Task



(b) Door Task

Figure 5-3: **Visual of constrained mechanisms tasks and parameterized policy spaces.** In both domains we assume the robot knows where the handle is, then explores an optimistic action space to determine which action parameterization maximizes reward. For the slider (a) the optimistic action space is the angle perpendicular to the plane normal (shown with a green vector), l, and the distance to slide the handle, z. For the door (b) the optimistic action space is the distance from the handle to the hinge, w, and the pitch of the frame attached to the door hinge, p, and the angle to open to door, q.

the true underlying reward function.

5.2 Constrained Mechanisms Domain

We demonstrate our method in a simulated domain containing prismatic and revolute joints with different kinematic parameters; an example of each is shown in Figure 5-3. The state is represented by an image, $s \in \mathcal{S} = \mathbb{R}^{wp \times hp \times 3}$ where wp and hp are the number of pixels in the width and height of the given image.

The objective of the slider task is to have the robot actuate a prismatic joint to its limit. The action space is the angle in which to slide the slider (about a vector normal to the slider plane centered at the slider handle), $l \in [0, \pi]$, and the distance to slide, $z \in [-0.25 \text{ m}, 0.25 \text{ m}]$, therefore $a = (l, z)$. We assume that the initial position of the slider handle and the normal to the slider plane are known. The reward is the distance that the robot is able to move the handle. The optimistic model for the slider task assumes that $\forall l \in [0, \pi]$ the slider will move in that direction the given distance z.

The objective of the door task is to open the door by actuating the revolute joint to its joint limit. The action space is the pitch of the frame attached to the door hinge where the y-axis is fixed and parallel to the door plane, $p \in [0, 2\pi]$, the distance from the hinge to the handle, $w \in [0.055, 0.15]$, and the opening angle of the door, $q \in [0, \frac{\pi}{2}]$, therefore $a = (p, w, q)$. We assume that the position of the door handle and the normal to the initial door plane are known. The reward is the distance that the robot is able to move the handle. The optimistic model for the doors task assumes that $\forall p \in [0, 2\pi]$ and $w \in [0.055, 0.15]$ the door handle will open to the given angle q.

In both tasks the reward is maximized by actuating the joint to its limit.

5.3 Implementation

We discuss the representation used for the task feasibility models in Section 5.3.1, then the details of the Gaussian Process in Section 5.3.2, and finally the PyBullet simulator we developed for our experiments in Section 5.3.3.

5.3.1 Task Feasibility Model Representation

We approximate the TFM, $\hat{R}_{task}$, with a neural network (NN) which consists of independent encoders for the input channels (images and action parameters) and a regressor which uses these encodings to predict reward. A visual of our architecture is given in Figure 5-4. The image encoder f_{im} has the form $z_{\text{im}} = f_{\text{im}}(s) = f_{\text{ss}}(f_{\text{cnn}}(s))$, where f_{cnn} is a Convolutional Neural Network (CNN). We found that mapping from the CNN directly into a fully connected layer did not result in useful encodings, so we added f_{ss}, which is a *spatial softmax* layer [25]. It generates, as output, a set of 2D feature points that are salient for making value predictions. Each 2D feature point is the expected pixel location of activations in one of the final CNN channels.

For the action inputs, a Multi-Layer Perceptron (MLP), f_{a}, transforms the action parameters into a latent space. This part of the network was designed so that additional action types could be added to the system, in which case, action parameters

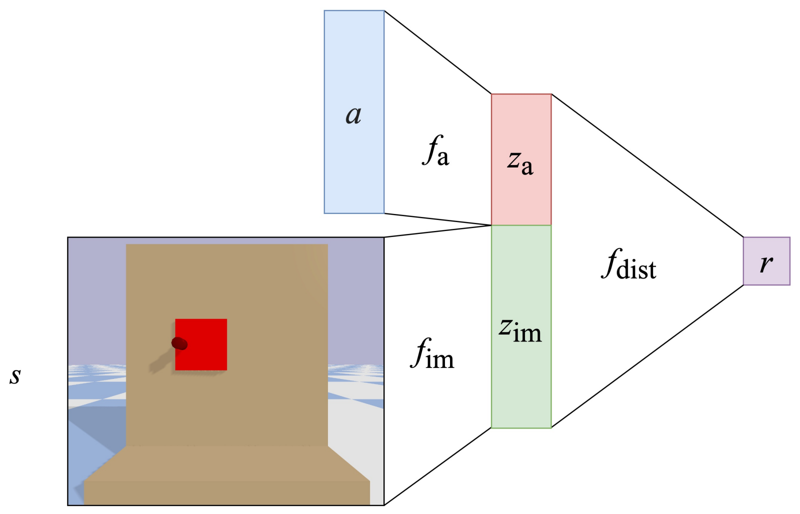


Figure 5-4: **The NN architecture for $\hat{R}_{task}$.** $\hat{R}_{task}$ predicts reward, r, given a state, s, represented as an image, and an action, a.

for all policies would be transformed into a shared space. This yields action encoding $z_a = f_{\text{a}}(a)$. Finally, z_{im} and z_a are concatenated and passed through an MLP, f_{dist}, which learns to predict how the action and image features map to a reward. The NN is composed of these encoders and the distance regressor as follows,

$$\hat{R}_{task}(s, a) = f_{\text{dist}}([f_{\text{im}}(s); f_{\text{a}}(a)]).$$

If $\hat{R}_{task}$ is trained to effectively approximate the true reward function R then an approximately optimal policy has the form

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{R}_{task}(s, a).$$

This formulation suffices for mechanisms that can be effectively actuated by a single relatively simple parameterized action. If we were to move to truly sequential mechanisms, such as gate latches, it would be necessary to treat the problem as a Markov Decision Process rather than a bandit.

5.3.2 Gaussian Process Details

To select actions according to the GP-UCB criteria given a state s, we must optimize the GP-UCB objective (Equation 4.2). To do this we start by evaluating the function on a coarse sampling of the action space. We then perform non-linear optimization on a few of the best samples, and select the best optimization run as the criteria-maximizing action. We found that using $\beta = 2$ in the GP-UCB objective (Equation 4.2) resulted in good optimization performance.

In our experimental results we use a squared exponential kernel and tune the kernel parameters by executing GP-UCB on a separate set of random mechanisms and observing the resulting exploration strategies. We aim to find a good balance between exploring areas of the input space with high uncertainty and areas with known high reward.

5.3.3 PyBullet Simulation

Using the PyBullet [18] simulator, we can generate multiple mechanisms where instances of the same joint type share visual structure. In our experiments, the robot will see a sequence of randomly generated mechanisms belonging to the same class. See Figure 5-3 for an example slider and door.

An action is a trajectory the end-effector should follow to actuate a joint with the corresponding action parameters. Given the pose of the mechanism handle, and action parameters, a trajectory is generated in the mechanism's configuration space. Then inverse-kinematics and Cartesian interpolation are used to generate a trajectory in Cartesian space. This trajectory is executed using a PD controller by applying forces to the handle. Due to the mechanism's constraints, the applied forces do not always result in motion.

5.4 Evaluation

In this section we describe the effectiveness of CPP in the slider and door domains. We find that CPP is able to learn from relatively few training mechanisms to quickly operate a new mechanism. We first discuss the regret metric used to evaluate CPP in Section 5.4.1, and describe the different methods which will be evaluated in Section 5.4.2. We give regret results in Section 5.4.3. We analyze different data generation methods and their performance in Section 5.4.4. Finally, we give a proof of concept of our method on a Baxter robot in Section 5.4.5.

5.4.1 Evaluation Metric

To measure the robot's success we assume that an oracle is able to provide the optimal reward, r^*, and we calculate the *normalized simple regret* (NSR) which is the loss of not selecting the optimal action

$$\text{NSR} = \frac{r^* - r}{r^*}, \tag{5.1}$$

where r is the reward for the robot's chosen action. For each new state presented to the robot during evaluation, we count the number of interactions needed until the robot can achieve $\text{NSR} < 0.05$.

5.4.2 Methods

The CPP method can work with any kind of exploration strategy during training time. We give results using two different exploration strategies during training time. We will discuss the comparison of these two methods in Section 5.4.4.

CPP-Random We randomly sample from the action space to collect training interactions for each mechanism.

CPP-GP-UCB We use the GP-UCB objective (Equation 4.2) to collect training interaction data. With this method the agent is actively trying to maximize its reward with each mechanism.

We compare our method against two simple but sensible baseline methods for evaluation that do not try to use previous experience and visual information about the new mechanism to predict how to actuate it. Thus, for the baseline methods, each mechanism is a new problem. While we expect performance to improve *within* one trial of interaction with a mechanism, we do not expect performance to improve *across* interactions with different mechanisms.

Random We randomly sample from the action space until the agent is able to maximize its reward.

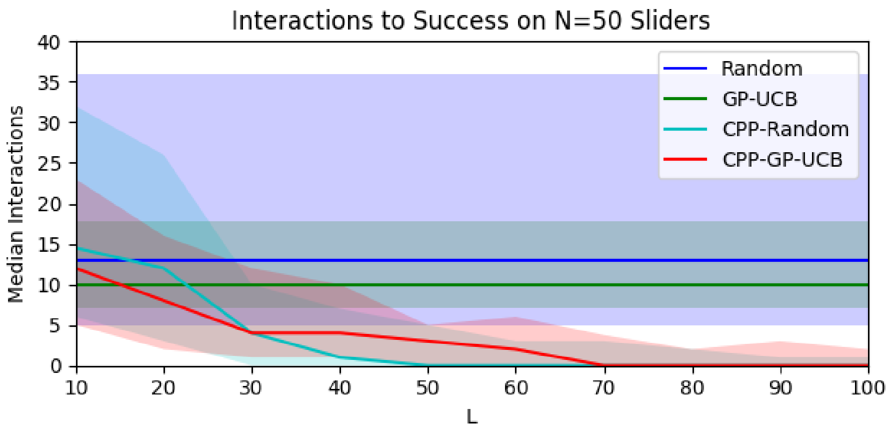
GP-UCB The robot uses the GP-UCB objective (Equation 4.2) for action selection until it is able to maximize its reward.

5.4.3 Regret Results

In both the baselines and in our evaluation of CPP, we limit the agent to 100 attempts at maximizing its reward. The results of our experiments are shown in Figure 5-5. Each plot shows the number of steps each method took to maximize the reward on the evaluation mechanisms as a function of L, the number of mechanisms the robot interacted with during training time. The baseline methods (blue and green) have the same median for all L values because they are not leveraging previous experience, and thus cannot show improvement.

The learning-based methods (red and cyan) show significant decreases in the number of interactions required to generate a successful interaction. Not only does the median number of interactions to success decrease with L, but so do the quantiles, meaning these methods more reliably interact with novel mechanisms. The larger L values are important to us: a well-trained robot is able to actuate a mechanism without any experimentation!

One drawback of CPP is that it can be misled by a poorly trained $\hat{R}_{task}$. In this case, the robot will first explore areas where $\hat{R}_{task}$ predicts high reward even if it is wrong. The GP-UCB algorithm will eventually correct the model's beliefs and explore other regions but this may take longer than an uninformed prior. We note that in all cases, as $\hat{R}_{task}$ starts performing better (after seeing more unique mechanisms), the



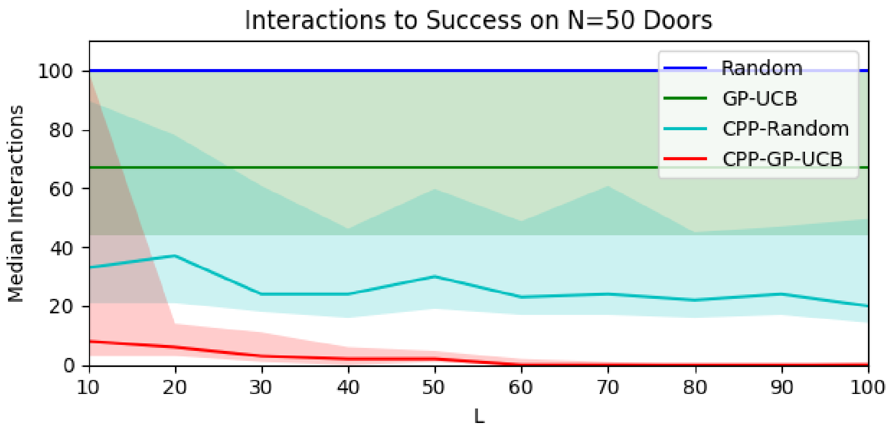


Figure 5-5: **Reward maximization results for Contextual Prior Prediction.** Number of interactions until success (less than 0.05 regret) on novel sliders (top) and doors (bottom). The median number of interactions is reported for 50 evaluation mechanisms for 5 separately trained $\hat{R}_{task}$ models. The plots show performance for models that have been previously trained on L mechanisms (x-axes) each with $M = 100$. We compare our method, noted as CPP-GP-UCB and CPP-Random, to GP-UCB which does not learn from previous interactions, and Random. 25% and 75% quantiles are plotted.

number of required interactions decreases.

To visualize the usefulness of CPP versus just trusting our $\hat{R}_{task}$ predictions, we compare an agent that simply selects the best action according to $\hat{R}_{task}$, to one that that uses our CPP method. Figure 5-6 shows a CPP agent which performs 10 GP-UCB interactions on top of the learned visual prior. As shown, CPP can still achieve low regret even with a poor $\hat{R}_{task}$ prior.

To visualize how the $\hat{R}_{task}$ prior improves over time, we show its predictions after

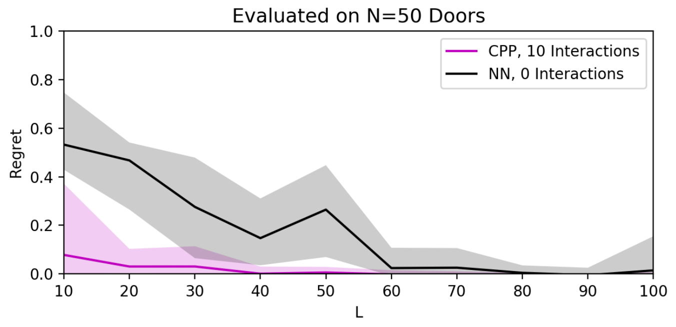


Figure 5-6: **Regret results for Contextual Prior Prediction versus directly optimizing $\hat{R}_{task}$.** This plot compares using $\hat{R}_{task}$ to directly predict optimal actions (NN), to using 10 GP-UCB interactions on top of $\hat{R}_{task}$ to find optimal actions (CPP). As shown, the NN initially results in poor performance as compared to CPP.

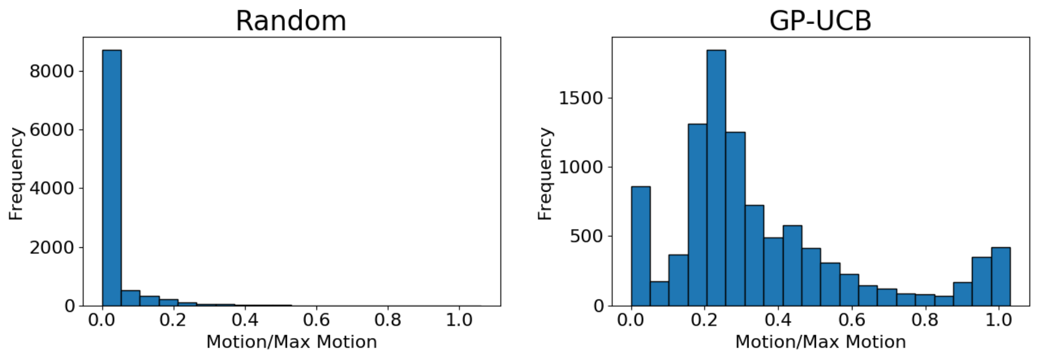


Figure 5-7: **Motion generated for different exploration strategies.** Motion histograms of the data collected for $M = 100$ steps on $L = 100$ doors using random data collection (left) and GP-UCB active data collection (right). The GP-UCB sampling is biased toward collecting samples with more motion.

being trained on an increasing number of sliders in Figure 5-8. The predictions get better at different rates for each slider which likely correlates to how similar the evaluation sliders are to the training sliders.

5.4.4 Data Generation

The primary utility of GP-UCB is to generate useful interactions during evaluation when the learned $\hat{R}_{task}$ does not have accurate predictions. We also experimented with using GP-UCB for collecting useful actions while interacting with training mechanisms.

Figure 5-5 shows that both the CPP-Random and CPP-GP-UCB methods perform

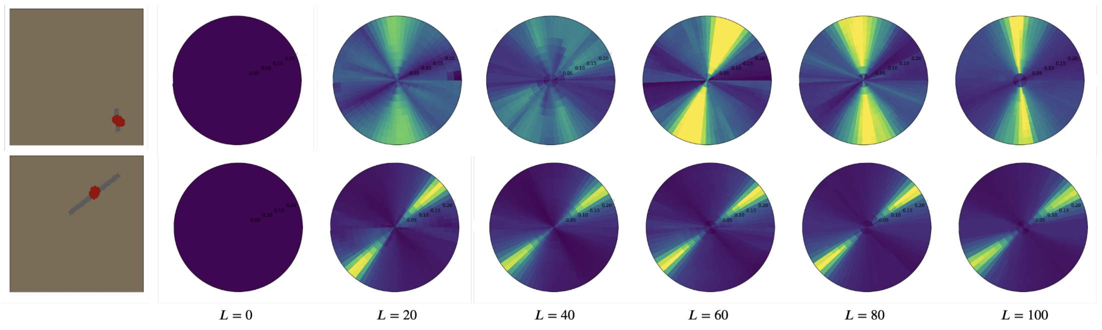


Figure 5-8: **Visualization of the $\hat{R}_{task}$ prior.** Prior visualized after experience with L previous sliders for 2 novel sliders. Each plot visualizes the predicted reward for an action, given in polar coordinates (direction and distance to move the handle). Yellow indicates a higher predicted distance. In all rows the predictions improve as L increases. However they all improve at different rates. This is most likely a factor of how closely these evaluation sliders correspond to sliders seen during training.

similarly in learning a good prior for slider mechanisms. However, for door mechanisms, we see that the CPP-GP-UCB method outperforms CPP-Random. This is due to the size of the action space, the size of the rewarding region of the action space relative to the size of the entire space, and the complexity of the reward function (how dependent the policy parameters are on each other). Figure 5-7 gives a histogram visualization of the training data used for door mechanisms. The random exploration strategy generates mostly zero motion, while the GP-UCB method is able to effectively actively explore the action space to find rewarding samples useful for training $\hat{R}_{task}$.

5.4.5 Real Robot Proof of Concept

We tested our learned model for slider mechanisms on a Baxter robot. The policy parameterizations are the same, and the trajectories output by the policies are fed into a position controller for the Baxter end effector, as shown in Figure 5-9. Our objective was to determine if the evaluation part of our pipeline could be executed on a real robotic platform. We observed that the Baxter was able to explore the real, novel slider mechanism when it started from both a poor (little previous experience) and good $\hat{R}_{task}$ model. The input image is a simulated version of the real mechanism

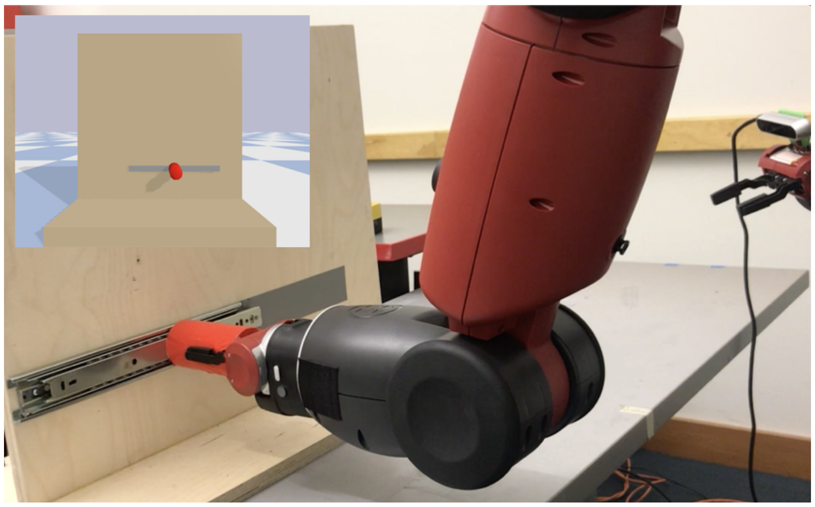


Figure 5-9: **Contextual Prior Prediction proof of concept with Baxter robot.** The real and simulated (inset) slider actuated by the Baxter.

as depicted in the inset of Figure 5-9. We surprisingly found that the Baxter was able to generate more motion for imperfect actions than the simulated agent. This was due to the compliance in the Baxter arm, which actually aided in shaping the reward, enabling the Baxter to learn the correct slider parameters with very few interactions.

5.5 Related Work

This work lies at the intersection of estimating kinematic models and policy learning. While the former can be used in the latter, there has been little work tying the two together with the goal of generalizing to novel objects through vision. Section 3.2 discussed related reinforcement learning techniques. Here we focus on other works related to the Contextual Prior Prediction method.

Of particular relevance to our work are other methods which estimate the kinematic parameters of articulated objects from interaction data [97, 8, 80, 52, 41, 23]. However, these works focus on learning the kinematic parameters for a single object. These works use visual representations such as optical flow [13, 52], or point features [23], which have been shown to be useful in predicting either kinematic models [52, 23] or rigid body separation [13]. Abbatematteo el al. [1] develop a method for predicting the kinematic parameters of a class of objects from an image. Similarly, we use a

CNN to learn visual features that are informative of motion in a class of objects.

A closely related problem is that of learning object affordances which is primarily concerned with the effects of actions as opposed to their rewards. In [77], the authors predict a probability distribution over possible object affordances from pixels. In [72], the authors learn the probability of successfully executing a grasp. Our method extends similar lines of work, in that the agent uses the learned model to seed on-line interactions.

GPs are useful for active exploration due to the fact that they give an estimate of the uncertainty over predictions [94]. However, using GPs requires careful specification of the kernel function which can be difficult to specify for images [51, 102]. In CPP we are able to reap the benefits of using a GP for exploration by using the image to initialize a GP over the low dimensional action space.

5.6 Conclusion

In this work, we focused on the problem of efficiently exploring novel mechanism instances by transferring knowledge from interactions with previous mechanisms through vision. We developed a method, Contextual Prior Prediction, that uses a learned prior mean for a GP and GP-UCB to maximize reward given a novel state. We evaluated our method in a continual C-MAB learning framework in a simulated domain consisting of prismatic and revolute joints, and proved that the evaluation strategies can be executed on a real robotic platform. We demonstrated that as the robot interacts with more mechanism instances, it can successfully actuate a new mechanism with an increasingly smaller number of interactions. Beginning with an optimistic action space enabled us to learn an accurate task feasibility model for predicting task reward. Future work needs to be done evaluating what would be necessary to learn relevant features for predicting motion from realistic images. In the next chapters we look into sequential manipulation domains and move away from using a pixel-based state representation in order to improve learning efficiency.

Chapter 6

Action Feasibility Model Learning

Long horizon sequential manipulation tasks still pose a challenging problem for robotic systems.[1] Tasks such as assembly depend on using many objects with varying physical properties. Finding a plan to achieve a task in these domains consists of reasoning over large spaces that include discrete action plans, as well as low-level continuous motion plans.

These problems can be effectively addressed hierarchically: at the highest level of abstraction the system searches over plausible *abstract* action sequences, and at the lower level it plans for detailed *concrete* motion plans and object interactions. The complexity of the search space for the concrete planner is greatly reduced when constrained by the abstract action sequence. Further computational efficiencies can be gained if we lazily [39] postpone concrete planning until we have a complete abstract plan that is likely to succeed, avoiding the need to query the concrete planner multiple times. A version of this approach is used in *skeleton-based* task and motion planning systems [68, 38, 55].

The success of this lazy strategy hinges on our ability to predict whether an abstract action sequence will be feasible to execute. In this work we leverage optimistic action models at the abstract level. We call an optimistic abstract action sequence

[1]The material in this chapter is based on:
Michael Noseworthy*, Caris Moses*, Isaiah Brand*, Sebastian Castro, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Active Learning of Abstract Plan Feasibility. Robotics: Science and Systems (RSS), 2021. [78]

feasible if both the concrete planner returns a solution and this solution is reliably executed in the real world with the intended outcome. The optimistic action model is a relaxation on the true dynamics and does not take into account errors in execution. As such the robot may end up attempting a plan that fails during execution.

Fortunately, even an approximately correct estimator of abstract action feasibility can offer huge computational advantages during planning. In some cases it may be possible to approximate action feasibility via coarse-grained simulation. However, this strategy still requires a coarse dynamics model, which may not capture complex phenomena needed to accurately predict feasibility in the real world.

Instead, we explore a strategy in which we learn an abstract action feasibility model (AFM) that predicts action feasibility by exploring the space of real plan executions without a specific planning problem or task at hand — a form of curious exploration [81]. Data efficiency is a primary concern in enabling real robot learning of feasibility models. Here, a training instance is the execution of an abstract action, labeled by success or failure. Labeling a sequence of abstract actions is very expensive as it involves finding and executing a concrete motion plan, potentially taking several minutes on a real robot. Furthermore, due to the combinatorial input space of abstract action plans, randomly executing actions is unlikely to elicit interesting behavior.

To address the data efficiency problem, we observe that in the process of training the AFM, some observations may be more valuable than others. Active learning is a technique for identifying unlabeled instances that are most informative in learning a target concept. The technical challenge is how to find plans of interest — an active learning approach requires both a way to generate candidate plans, and a way to score how informative a candidate plan might be given the current model.

To determine how informative a plan is with respect to the learned AFM, we adopt an information-theoretic active learning approach [70, 45]. To generate candidate plans, we exploit an important property of abstract action sequences: for an action sequence $a_{1:N}$, if any prefix $(a_1, \dots, a_i)$ is infeasible, then any longer prefix $(a_1, \dots, a_j)$ for $i < j \leq N$ is also infeasible. This *infeasible subsequence property* gives us leverage during data acquisition. A complex plan instance may contain many elements that

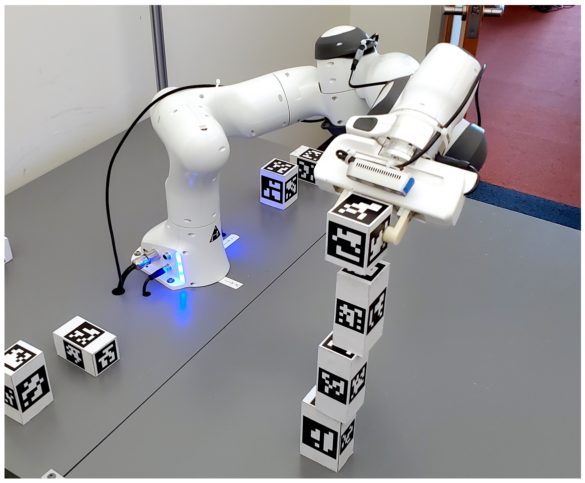


Figure 6-1: **Panda tower construction.** This shows the Franka Emika Panda robot constructing a tower to improve its understanding of action feasibility. A wrist-mounted camera refines object pose estimates for precise grasping. Unique ArUco markers [34] are applied to each face of each object for object identification and localization.

are highly informative for model learning, but will never be experienced because early elements in the plan will fail with high probability.

We apply this active learning strategy to the concrete problem of stacking blocks with a real robot, where the blocks are each unique and have non-uniform mass distributions. The robot autonomously designs, plans, and executes experiments to learn action feasibility models using a Franka Emika Panda robot arm (Figure 6-1). The robot is also capable of resetting the world state after each experiment, enabling continuous autonomous experimentation. The learned feasibility predictor is later used to build towers with previously unseen blocks that satisfy several different objective functions, including the tallest possible tower or the tower with the longest overhang. This sample-efficient autonomous learning process relieves engineers from supervising data collection, resetting the experimental environment, and having to specify accurate dynamics models for planning. This results in a highly flexible and robust system for planning and executing complex action sequences in the real world.

In summary, our contributions are:

- A method to learn an abstract action feasibility model (AFM) by synthesizing hypothetical optimistic plans;

- A data acquisition approach which leverages the *infeasible subsequence property* when sampling potential plans;
- A robotic system which conducts autonomous self-supervised learning via integrated perception, experimentation, planning, and execution.

6.1 Method

Here we give a method for solving the action feasibility model (AFM) learning problem outlined in Section 2.3.1. Specifically, we want to learn an AFM which can be used with a planner to maximize a given task reward,

$$a_{1:N} = \underset{a_{1:N} \in P}{\operatorname{argmax}} \, \text{PF}(s_0, a_{1:N}) R_{task}(s_N),$$

where plan feasibility, or PF (Equation 2.1), depends on the learned AFM. We take an information-theoretic approach to actively learning an AFM. An overview of the learning and evaluation-phases are given in Figure 6-2. First we give our method, Sequential Actions, in Section 6.1.1, then we discuss the representation of the AFM in Section 6.1.2.

6.1.1 Sequential Actions

Our method builds upon Bayesian Active Learning by Disagreement (BALD) [45], described in Section 4.2. As discussed, the BALD objective's (Equation 4.4) derivation relies on the underlying information gain problem being submodular. Informally, a submodular problem is one in which executing any one action does not have the potential to make future actions more informative. We make the observation that information gain in block stacking is *not* submodular on the level of individual block placement actions, but *is* submodular on the level of towers constructed. Building a single tower will always give us information, and that amount of information strictly decreases over time (i.e. has diminishing returns). However *within* a single tower, certain block placements can give more or less information depending on where and

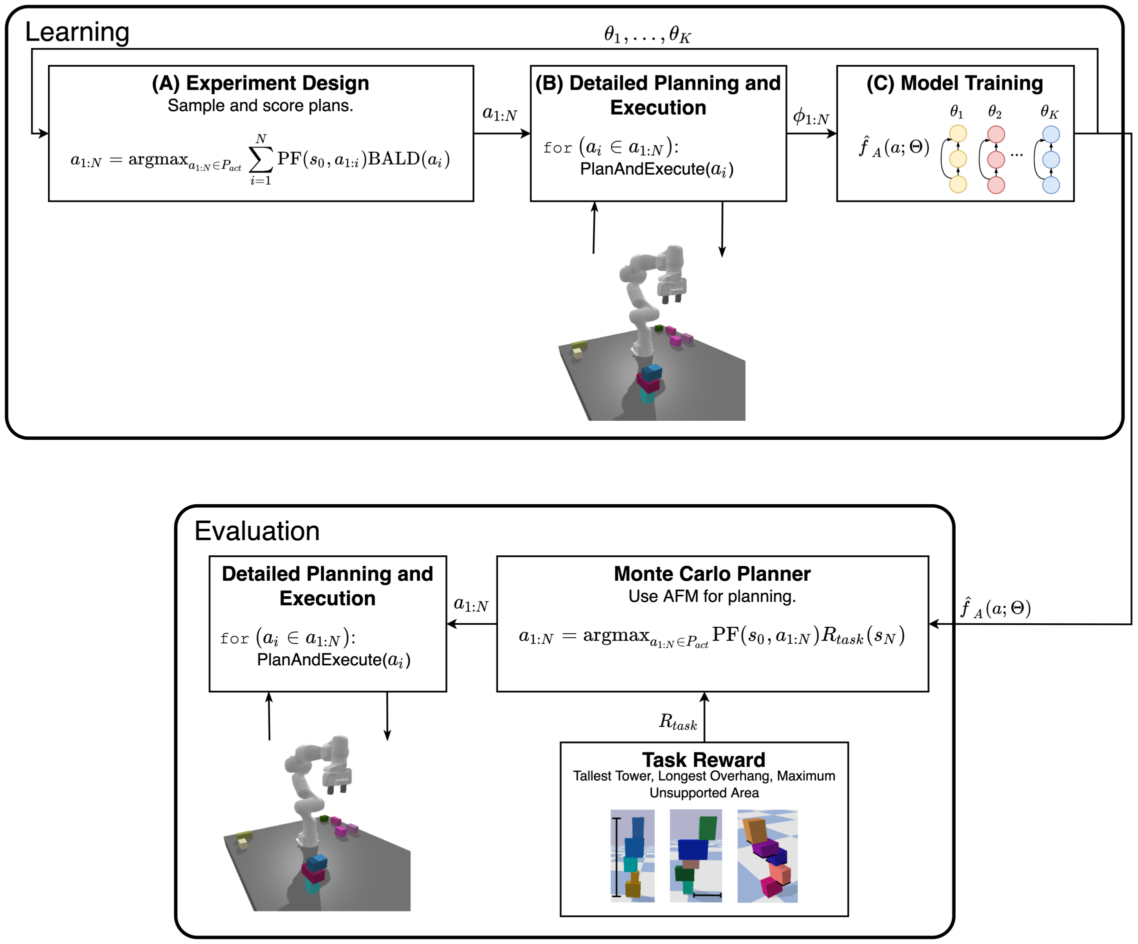

Learning
$\theta_1, \ldots, \theta_K$
(A) Experiment Design
Sample and score plans.
$a_{1:N} = \text{argmax}_{a_{1:N} \in P_{act}} \sum_{i=1}^{N} \text{PF}(s_0, a_{1:i})\text{BALD}(a_i)$
$a_{1:N}$
(B) Detailed Planning and Execution
for $(a_i \in a_{1:N})$: PlanAndExecute(a_i)
$\phi_{1:N}$
(C) Model Training
$\hat{f}_A(a; \Theta)$
θ_1 θ_2 ... θ_K
Evaluation
Detailed Planning and Execution
for $(a_i \in a_{1:N})$: PlanAndExecute(a_i)
$a_{1:N}$
Monte Carlo Planner
Use AFM for planning.
$a_{1:N} = \text{argmax}_{a_{1:N} \in P_{act}} \text{PF}(s_0, a_{1:N}) R_{task}(s_N)$
$\hat{f}_A(a; \Theta)$
R_{task}
Task Reward
Tallest Tower, Longest Overhang, Maximum Unsupported Area

Figure 6-2: **Learning and evaluation-phase for action feasibility models.** The proposed system for learning an abstract AFM operates in two phases. **Learning-phase** (top) The robot iteratively designs and executes experiments that improve its AFM. **(A)** Using its current model, the robot selects the abstract action sequence, $a_{1:N}$, that maximizes information gain over the AFM. The Sequential Actions objective is shown here, but it can be replaced with any of the baseline active learning strategies given in Section 6.4.1. **(B)** The robot then computes and executes a concrete motion plan for $a_{1:N}$. **(C)** After observing the true action feasibilities, the robot uses this new labeled data to update its AFM, represented by an ensemble of neural networks. **Evaluation-phase** (bottom) Once an AFM has been learned the robot can use it to perform various tasks, such as building the tower with the longest overhang from a given novel set of blocks.

when they are stacked. One could imagine setting up a tower building experiment with the intent of gaining the most information on the last block placement in the sequence, proving that information gain on the level of block placements does not always have diminishing returns.

When calculating the BALD objective (Equation 4.4) over plans (towers), the input is a sequence of actions, $x = a_{1:N}$. The output is a label of whether or not that tower was stable after construction, $y = \phi$. However, if we generate candidate plans using the optimistic model, f_O, then we could end up selecting a plan which fails to execute all intended actions. Using the BALD objective in this way does not account for the fact that some of the actions in the sequence $a_{1:N}$ may never be experienced due to a previous action in the sequence being infeasible. We refer to this as the *infeasible subsequence property*. In sequential domains in which reaching parts of the state space are dependent on previous actions taken, it is important to consider the feasibility of an action given the actions taken up to that point.

We therefore reformulate the objective by using our plan feasibility (PF) calculation (Equation 2.1). We also change the BALD objective to now take individual actions as input $x = a$ and individual action labels as output, $y = \phi$. By combining plan feasibility and information gain over individual actions, we maximize information gain over *all* actions in the sequence by considering our ability to actually execute each action to acquire each label. This gives the Sequential objective

$$\underset{a_{1:N} \in P}{\operatorname{argmax}} \sum_{i=1}^{N} \text{PF}(a_{1:i}) \text{BALD}(a_i). \tag{6.1}$$

When $P = P_{act}$, which consists of plans generated by random roll-outs of the optimistic transition model, f_O, we refer to it as the Sequential Actions objective.

In our stacking domain, the state and action are both represented by an abstract action a. Therefore the AFM operates only on action a to predict feasibility, $\hat{f}_A(a; \Theta) \to [0, 1]$. We will discuss the details of the combined state action representation for the stacking domain in Section 6.2.1.

6.1.2 Action Feasibility Model Representation

To deal with domains in which the state representation has variable length, we leverage graph neural networks (GNNs). GNNs make predictions based on aggregations of local properties and relations among the input entities, and exploit parameter tying to model global properties of plans of arbitrary size using a fixed-dimensional parameterization Θ. Many recent works have focused on learning predictive dynamics models. In such scenarios, it has been shown that explicitly representing objects and their relations in the model can lead to more efficient learning and generalization [9, 10, 16, 107]. As such, graph networks are becoming a more common modeling choice in these domains [55, 88]. For a detailed description of GNNs, see the overview by [109].

The active learning objective is to maximally reduce the entropy of $\Pr(\Theta \mid \mathcal{D})$. In general, for complex model classes such as neural networks, an explicit representation of $\Pr(\Theta \mid \mathcal{D})$, the distribution over the parameter space given training data $\mathcal{D}$, is difficult to construct or update with new data. We therefore follow the strategy of Beluch et al. [12] and represent uncertainty with an ensemble of individual neural network models. For each individual model, initial parameters are drawn independently at random and are updated to incorporate new data via gradient descent in different orderings of the dataset.

Using an ensemble of equally weighted parameter vectors $(\theta_1, \ldots, \theta_K)$ to represent $\hat{f}_A(a; \Theta)$ allows us to compute a global feasibility prediction,

$$\hat{f}_A(a; \Theta) = \frac{1}{K} \sum_{i=1}^{K} \hat{f}_A(a; \theta_i).$$

We find an experiment that maximizes the estimated BALD objective using an ensemble with the following calculation

$$\text{BALD}(a) = \mathcal{H}[\hat{f}_A(a; \Theta)] - \frac{1}{K} \sum_{i=1}^{K} \mathcal{H}[\hat{f}_A(a; \theta_i)].$$

In Section 6.3.1 we give the details of the GNN used in the stacking domain.

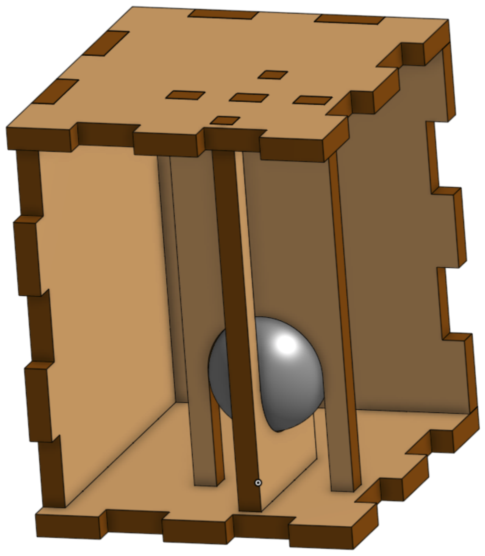


Figure 6-3: **Non-uniform mass distribution blocks.** The cuboids for the real robot experiments were constructed from laser cut plywood. A 25mm diameter lead ball is mounted randomly inside some of the objects to significantly alter the mass distribution.

6.2 Stacking Domain

We implement this framework for a class of problems in which the robot manipulates objects to construct towers. All of our experiments use the 7-DOF Panda robot from Franka Emika either in a simulated PyBullet environment or in the real world.

6.2.1 State and Abstract Action Representation

The world consists of the robot and a set of objects, $\mathcal{O}$, with which it can interact. In this work, we consider cuboids with non-uniform mass distributions (Figure 6-3). Each object, $o \in \mathcal{O}$, is described by a tuple, (d, c, m), where $d \in \mathbb{R}^3$ are the dimensions, $c \in \mathbb{R}^3$ is the offset of the center of mass from the center of geometry, and $m \in \mathbb{R}$ is the object's mass.

During the learning-phase we use a set of 10 blocks, and all evaluations are performed with a different set of 10 blocks. The block parameters from each set are sampled from the same uniform distribution over the dimensions of the objects and the locations of the center of mass within the objects.

In this domain we combine the state and abstract action into a single action representation. An abstract action is specified by $a = (\mathbf{o}, \mathbf{r})$ where $\mathbf{r} \in SE(3)$ are the relative poses of all objects $\mathbf{o}$ with respect to the objects below them (or the table if this is the first object in the list). The action portion of this representation is getting the final object to the final relative pose. The robot can plan in this abstract action space, and consider abstract actions that are infeasible to construct. However, when constructing and labeling abstract actions to learn $\hat{f}_A$, we only label abstract actions which either succeed or fail on the last block placement. Once the tower falls we do not label any future abstract actions that the robot was considering.

This abstract action is optimistic in that it believes that any contact between the block being placed on the tower and the block below it will result in a stable tower. However this will not always be the case due to the effects of the center of mass not being considered, as well as any noise in the underlying robot system.

An abstract action a is labeled as feasible, $\phi = 1$, if the last block placement results in the tower remaining stable, and infeasible, $\phi = 0$, otherwise. Feasibility in this scenario equates to the robot being able to construct a specific tower, so it corresponds to *constructability* as opposed to *stability*. A tower can be stable but not constructable, but all constructable towers are stable towers. In this chapter we use the term stable as it is more intuitive, but plan feasibility is actually tower constructability.

6.2.2 Low-level PDDL Action Models

We use PDDLStream [33] (discussed in Section 4.3.1) to find the concrete plan, consisting of grasps and motion plans, needed to execute an abstract action. In this section we given the predicates and PDDL action models used to find a concrete plan for a corresponding abstract action a.

Predicates

The parameters for our actions and predicates are the following: `?o` is an object (either the table or a block), `?p` is an object pose, `?g` is a grasp, `?q` is a robot arm configuration, and `?t` is a robot arm trajectory in joint space.

The predicates `AtPose`, `AtConf`, and `HandEmpty` model the changing pose of objects, configuration of the robot arm, and state of the robot gripper. When the robot is holding something, the grasp is modeled with `AtGrasp`. `Block` and `Table` simply mean that the given object is a block or table, respectively. `On` indicates that `?o1` is resting stable on top of `?o2`. `PickKin` (`PlaceKin`) indicates that object `?o` can be picked (placed) from (at) `?p` with grasp `?g` following trajectory `?t` from configuration `?q1` to `?q2`.

`FreeMotion` indicates that trajectory `?t` from configuration `?q1` to `?q2`, where the robot is not holding an object, is collision-free. `HoldingMotion` indicates that trajectory `?t` from configuration `?q1` to `?q2`, where the robot is holding object `?o` with grasp `?g`, is collision-free.

`Stackable` is a derived predicate which indicates that object `?o1` can have something stacked on top of it if it is either a table, or a block with nothing on top of it.

```
(:derived (Stackable ?o1)
  (or (forall (?o2) (not (On ?o2 ?o1))) (Table ?o1)))
```

`Moveable` is a derived predicate which indicates that object `?o1` is a block and can be moved (has nothing on top of it).

```
(:derived (Movable ?o1)
  (and (Block ?o1) (not (exists (?o2) (On ?o2 ?o1)))))
```

Action Descriptions

The following `MoveFree` and `MoveHolding` action descriptions model the conditions under which the robot can move from one configuration to another while either not holding a block, or while holding a block.

```
(:action MoveFree
    :param (?q1 ?q2 ?t)
    :pre (and (FreeMotion ?q1 ?q2 ?t)
              (AtConf ?q1)
              (HandEmpty))
    :effect (and (AtConf ?q2)
                 (not (AtConf ?q1))))

(:action MoveHolding
    :param (?o ?g ?q1 ?q2 ?t)
    :pre (and (HoldingMotion ?o ?g ?q1 ?q2 ?t)
              (AtConf ?q1)
              (AtGrasp ?o ?g))
    :effect (and (AtConf ?q2)
                 (not (AtConf ?q1))))
```

The following `Pick` and `Place` action descriptions model the conditions under which the robot can either pick up or place an object.

```
(:action Pick
  :param (?o1 ?p1 ?o2 ?g ?q1 ?q2 ?t)
  :pre (and (PickKin ?o1 ?p1 ?g ?q1 ?q2 ?t)
            (AtPose ?o1 ?p1)
            (HandEmpty)
            (AtConf ?q1)
            (Movable ?o1)
            (On ?o1 ?o2))
  :effect (and (AtGrasp ?o1 ?g)
               (AtConf ?q2)
               (not (AtConf ?q1))
               (not (AtPose ?o1 ?p1))
               (not (HandEmpty))
               (not (On ?o1 ?o2))))
```

```
(:action Place
    :param (?o1 ?p1 ?o2 ?g ?q1 ?q2 ?t)
    :pre (and (PlaceKin ?o1 ?p1 ?g ?q1 ?q2 ?t)
              (AtGrasp ?o1 ?g)
              (AtConf ?q1)
              (Stackable ?o2))
    :effect (and (AtPose ?o1 ?p1)
                 (HandEmpty)
                 (AtConf ?q2)
                 (not (AtConf ?q1))
                 (not (AtGrasp ?o1 ?g))
                 (On ?o1 ?o2)))
```

6.2.3 Task Reward Functions

The learned AFM model is applied to three different objectives in the evaluation-phase:

1. *Tallest Tower:* The objective is to construct the tallest possible tower.

2. *Longest Overhang:* The objective is to construct the tower with the maximum distance from the center of geometry of the bottom block to the furthest vertical side of the top block.

3. *Maximum Unsupported Area:* The objective is to construct the tower where each block has as much area as possible unsupported by the block below it.

6.3 Implementation

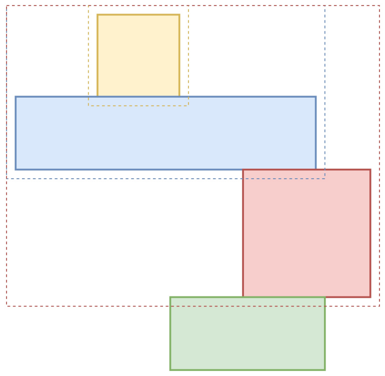
We discuss the details of the ensemble of GNNs used to represent the learned AFMs in Section 6.3.1. Then, we discuss the details of our planner in Section 6.3.2. In Section 6.3.3 we describe the perception system used to localize blocks in our real world experiments, and in Section 6.3.4 we give our real world experimentation setup. Finally, in Section 6.3.5 we explain how our method could be applied to other domains.

6.3.1 Learning

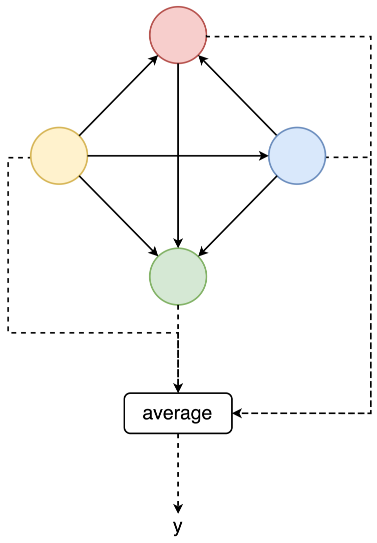
The distribution over AFM model parameters is represented by an ensemble of networks. In our implementation, the ensemble is made up of GNNs with domain-specific connectivity. The input to the GNN is an abstract action, a. Each block placement in a is represented by a separate node in the graph, and each node is connected to nodes above it in the tower. This mirrors the analytical computation of tower stability. If we consider each subtower in the tower that starts with the top block, they must all have a combined center of mass within the contact patch of the block below them. This is shown visually in Figure 6-4. We refer to this network architecture as a Towers Graph Network (TGN). Other network architectures which are invariant to task plan length are a Fully Connected Graph Network (FCGN) and a LSTM model. The FCGN uses the same node and edge networks as our TGN model, but has edges between each node. The LSTM model passes each block's vector representation through the network in order starting from the top of the tower, and most closely matches our TGN connectivity. A visualization of the connectivity of each architecture is given in Figure 6-5.

In our experiments, 10 networks are used in the ensemble. Each individual network is randomly initialized and trained using the binary cross-entropy loss function with early stopping according to the loss on a validation set, which is also collected actively. We also perform data augmentation by rotating each collected tower $90, 180, \text{and}, 270$ degrees about the axis normal to the table surface.

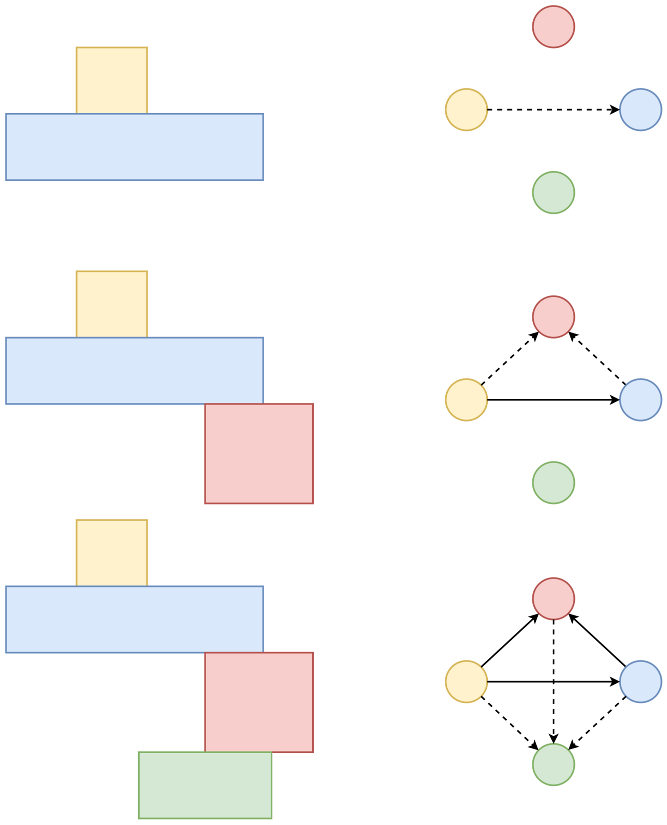
Before active learning, each model in the ensemble is initialized by training on the same dataset (shuffled differently for each) of 40 randomly generated plans of length 2. During the learning-phase, at each iteration the top 10 most informative plans are chosen and labeled by attempting to execute each with the robot. 20% of the collected data is added to a validation set and the remainder is added to the training set.



(a) Tower and Subtowers



(b) TGN Predication



(c) TGN Connectivity

Figure 6-4: **Towers Graph Network and analytical stability calculation.** Consider the tower shown in (a). Calculating the stability of this tower consists of calculating the combined center of mass of each subtower, and determining if it is in the contact patch between the bottom-most block in the subtower, and the block supporting the subtower. In this example that would consist of calculating the joint center of mass of the subtowers highlighted in yellow, blue, and red, and seeing if each subtower's joint center of mass is within the contact patch between the yellow and blue block, the blue and red block, and the red and green block, respectively. (b) shows how the outputs from each node in the graph network are averaged to get the final network prediction. (c) shows how each subtower is used to determine the connectivity of the TGN for this given tower.

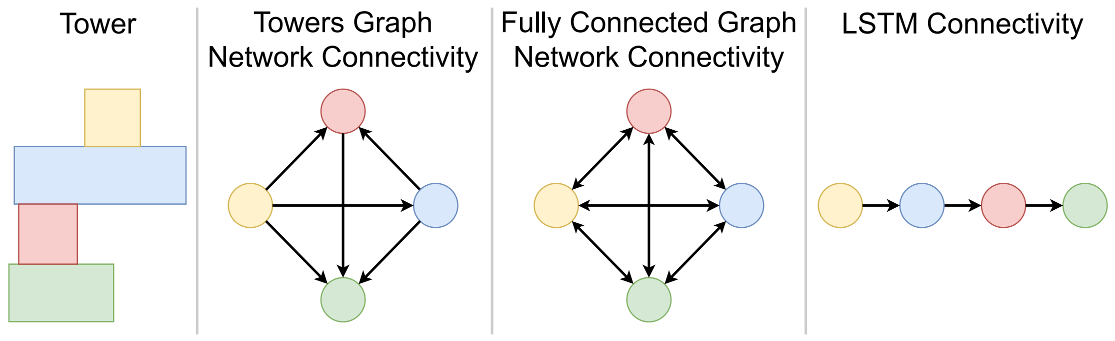

Tower
Towers Graph Network Connectivity
Fully Connected Graph Network Connectivity
LSTM Connectivity

Figure 6-5: **Different network architectures for learning action feasibility models in the towers domain.** The flow of information for the model architectures compared in Section 6.4.4. The TGN uses domain specific connectivity, while the FCGN assumes no prior knowledge on how the blocks should be used together to inform predicting feasibility. The LSTM simply iterates through the blocks starting with the top block.

6.3.2 Planning

Planning consists of first finding a high-level plan of abstract actions, then finding a consistent low-level plan with the motions and grasps necessary to achieve the high-level objectives. In this domain, abstract actions are simply the poses of all blocks in a tower. To continuously execute abstract action plans, the low-level planner must consider many scenarios, such as regrasps and moving fallen blocks back to their home positions before building a new tower. First we discuss the high-level Monte Carlo planner, then the low-level planner.

High-Level Planner

At the high-level we use a Monte Carlo planner which simply rolls out random abstract actions, of a given length N, to generate plan space P_{act}. During the learning-phase we score plans in P_{act} using the Sequential Actions objective. During the evaluation-phase, we score plans using Equation 6.1, a combination of learned plan feasibility and the task reward.

Low-Level Planner

To handle low-level planning, we use the PDDL action models given in Section 6.2.2 with PDDLStream [33] (described in Section 4.3.1). This planner handles task level reasoning and uses a Bidirectional RRT [58] in joint configuration space to perform motion planning and collision checking with a surrogate world model implemented in PyBullet [18]. In this work we make the simplifying assumption that there are dedicated positions for the base of the tower, regrasping, and storage for each of the objects. These constraints are specified to the planner to reduce planning time by limiting the search space of possible action parameters.

To make planning more efficient, we break up calls to PDDLStream by passing in a single desired block pose at a time. We iterate through each abstract action, plan to achieve the last block placement using the low-level PDDLStream planner, then execute the concrete plan. For each block placement, we give the planner a goal state of `(Pose ?o ?p)`, where we calculate the desired pose `?p` of block `?o` to be placed from the abstract action a. After executing the concrete plan for each abstract action, we check whether or not the tower has fallen. If it is still stable we continue planning and executing block placements. If the tower falls during execution or the entire tower is successfully constructed, then we pass new goals to the planner which consist of getting each block back to its home position starting with the block highest up in the z-direction (normal to the table). This heuristic, of starting with the top block, is helpful when deconstructing a stable tower. It is also helpful when the blocks have fallen into a pile, allowing the robot to deconstruct the pile starting from the top.

6.3.3 Perception

We perform experiments in both a simulated PyBullet environment as well as a real robotic environment. In simulation we perceive blocks using their ground truth poses. On the real platform we engineer a perception system. For the system to robustly pick up blocks and recover from unstable towers falling in unpredictable configurations,

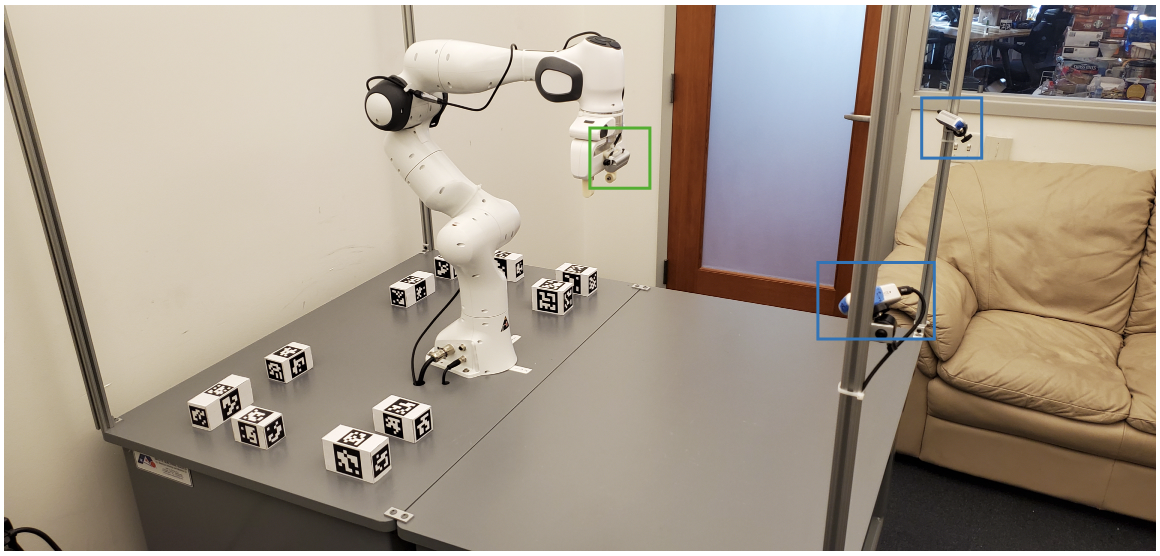


Figure 6-6: **Panda stacking domain setup.** Our setup includes two world-view cameras, shown in blue. A wrist-mounted camera, shown in green, is used for more accurate grasping. All blocks are shown in their home positions.

we require a perception system that can identify and localize objects at arbitrary positions. Although more advanced perception systems might be needed for arbitrary objects, vision is not the immediate focus of this work, so we pattern our objects with ArUco markers [34] to simplify perception. To indicate identity and avoid orientation ambiguity, each object has a unique ArUco marker on each face.

Two RealSense D435 depth cameras (shown in blue in Figure 6-6) mounted statically on a frame observe the workspace and allow for localizing the objects with minimal occlusions. If an object is not visible, it is assumed to be at its home position behind the arm. Due to the resolution of the cameras and size constraints of the tags on the blocks, we found that the pose estimates from the static cameras can have up to 1 centimeter of error. This level of error is acceptable, as rough pose estimates are refined with a third RealSense D435 camera mounted on the robot wrist (shown in green in Figure 6-6). As the arm moves to a pre-grasp pose computed from the noisy object pose estimate, the wrist-mounted camera collects images closer to the the object to be grasped, allowing for a refined pose estimate and more precise grasp.

6.3.4 Execution

In simulation, motion plans and grasps are executed by moving the robot through joint configuration trajectories. On the real robot, motion plans are executed using joint-space controllers on the robot. When constructing a tower in the learning-phase, after each block placement, the wrist camera is used to check for tower stability. If a tower was unstable, then the last manipulated block will not be near its expected pose in front of the gripper.

To improve data collection efficiency, we parallelize execution and planning. As the robot executes a motion plan to assemble or disassemble a tower, the planner produces plans to move each individual block in that tower under the assumption that all actions will be successful. This parallelism is interrupted if a tower falls over prematurely, prompting a replan to clear the fallen blocks.

If the state of the world is such that a robot is unable to find a plan to proceed with experimentation or execute an existing plan, human intervention may be required. We have provisioned for several of these cases, including blocks falling off the table, or too close to one another for the planner to find feasible grasp candidates. Once such issues are manually resolved (e.g., by putting the block back on the table or pushing them apart from each other), the robot can update its estimate of block poses and resume planning.

6.3.5 Method Generalizability

Here, we motivate a more general class of problems for which our system applies and clarify which components of the system are specific to our chosen domain. Our method most benefits domains where the feasibility of an action depends strongly on the preceding action sequence (e.g., adding a fifth block to a tower that already has four blocks). Domains that include construction tasks, like ours, will commonly benefit from non-myopic information gathering and a feasibility predictor that incorporates previous actions into its predictions. Consider a packing problem where many objects need to be placed in a larger container without any becoming damaged. To apply our method to a new domain the overall system/methodology would remain unchanged. However, one would need to adapt the following domain-specific components:

1. *Optimistic action definitions.* Parameterize an optimistic action that is appropriate for the domain and connect it to concrete actions (e.g., object locations within the container). Ensure that the feasible subspace of the action (actions which can feasibly be placed without causing damage) is within the space of the optimistic model's predictions.

2. *Experimental infrastructure.* Additional capabilities to autonomously plan and execute optimistic actions in the physical world (e.g., motion and grasp planning infrastructure).

3. *Feasibility detector.* For a new action, we require a method to autonomously acquire the feasibility label during execution (e.g., whether the gripper will collide when placing the object or if an object will be damaged).

4. *Additional inductive bias* (optional). Additional structure to the learner can further increase data efficiency, as shown by our TGN method. However, this is not required as we show in Figure 6-5 that a general purpose graph network can be used (e.g., a graph network that has connectivity between all objects).

In Chapter 7 we apply learning AFMs to a tool-use domain and give many more details on how this method could be adapted to a new domain.

6.4 Evaluation

First in Section 6.4.1 we discuss the baselines we compare our Sequential Actions method to. In Sections 6.4.2 and 6.4.3 we evaluate the learning efficiency and task performance of the different active learning strategies. In Section 6.4.4 we analyze the ability of the learned AFMs to generalize, as well as how different NN architectures can impact the accuracy of the learned AFMs. These results are generated in a simulated towers domain. Finally, Section 6.4.5 gives the performance of learning an

AFM on a real robot. We show that not only can the robot learn an AFM from real data and use it to perform downstream tasks, but also that learning on the real robot allows us to be robust to system noise.

When assessing task performance during the evaluation-phase, we randomly select $N = 5$ blocks from the set of 10 novel evaluation blocks and execute the best tower (that uses all 5 blocks) found by the Monte Carlo planner, given the task objective and our AFM. The reward received from executing this tower is used to calculate normalized regret (Equation 5.1), which is the difference between this received reward and the maximum possible reward. If a tower is unstable, we assign a reward of zero. We calculate the maximum possible reward by considering each tower evaluated by the Monte Carlo planner within a simulator with no added noise, and selecting the one which both maximizes reward and is stable.

6.4.1 Baselines

We compare to several baselines. First we discuss methods which learn an AFM, then we discuss methods which learn a full plan feasibility model.

Action Feasibility Model Baselines

In all methods described in this section, the robot receives a feasibility label after each action is executed. What varies is how actions are selected.

Incremental The Incremental strategy leverages the *infeasible sub-sequence property.* In this method we only consider plans $a_{1:N}$ for which we have already observed the prefix to be feasible. In other words, the prefix $a_{1:N-1}$ is in the current dataset with labels $\phi_i = 1$ for $1 < i < N - 1$.

$$\underset{a_{1:N} \in P_{inc}}{\text{argmax}} \, \mathbb{1}_{\{(a_i, \phi_i = 1) \text{ for } 1 < i < N-1 \in \mathcal{D}\}} \text{BALD}(a_N)$$

We generate P_{inc} by randomly stacking a block on sequences of actions which are already in the dataset with positive labels.

Greedy Actions The Greedy Actions approach selects the next action a_N which maximizes the BALD objective, given that we have already constructed $a_{1:N-1}$ and the tower is still stable. This strategy does not take into consideration that if a plan fails early, we do not get to learn from the full plan execution, which is the *infeasible sub-sequence property.*

Random Actions The Random Actions approach selects the next action, a_N, randomly, given that we have already randomly executed $a_{1:N-1}$ and the tower is still stable. It also does not take into account the *infeasible sub-sequence property.*

Plan Feasibility Model Baselines

We also compare to baselines which model abstract *plan* feasibility, as opposed to abstract *action* feasibility, These methods do not account for the *infeasible sub-sequence property.* In the previously discussed AFM learning methods, we are able to use our learned models to calculate plan feasibility, PF (Equation 2.1). However, now we directly represent a plan feasibility model which has no notion of individual action feasibility. We are no longer learning an action feasibility model, $\hat{f}_A : \mathcal{A} \rightarrow \mathbb{R}$, but a plan feasibility model, $\hat{f}_{PLAN} : \mathcal{A}^N \rightarrow \mathbb{R}$. In this setup the robot only receives labels for each tower as opposed to each action. If the full tower is constructed without falling it receives a positive label, $\phi = 1$, and if it falls during construction it receives a negative label, $\phi = 0$. A dataset for learning $\hat{f}_{PLAN}$ consists of $(a_{1:N}, \phi)$ where a single label corresponds to each plan (tower) in the dataset.

Complete The Complete strategy uses the BALD objective with the learned $\hat{f}_{PLAN}$ to select a plan which maximizes information gain.

Random Plans The Random Plans strategy plans by rolling out random sequences of optimistic actions, then executes the generated plan.

6.4.2 Impact of Action Feasibility Model Learning Strategy on Task Performance

Figure 6-7 shows the task performance of models trained using our Sequential Actions method against baselines which learn an AFM. Each strategy has results aggregated from 4 independent training runs and 50 task evaluations per run all performed in a simulated environment. Different sets of blocks are used between training and evaluation.

Our AFM performs best on the *Tallest Tower* task, reaching a median regret of roughly 0 after constructing only 200 towers with the Sequential Actions method. The Incremental method also performs well, successfully minimizing regret with minimal variance across runs. Note that the shaded region represents the quartile distribution — a region that extends to 1.0 means that more than a quarter of the trials were unstable. This highlights the importance of considering the *infeasible sub-sequence property* when sampling and scoring plans in the action space. Finally, the naive Greedy Actions strategy performs the worst, and is only able to achieve decent performance on the *Tallest Tower* task after seeing roughly 800 training towers, likely due to the fact that it is not considering the feasibility of subtowers when searching the actions space, just greedy single-block placements.

The *Maximum Unsupported Area* and *Longest Overhang* tasks are more challenging for the robot because they require deep understanding of the tower stability decision boundary, while the *Tallest Tower* task only requires a rough understanding of how to build stable towers with high confidence. These results show that in spite of the difficulty of the first two tasks, the active learner is able to improve its understanding of the decision boundary well enough to perform tasks with very low regret and low variance.

6.4.3 Impact of Plan Feasibility Model Learning Strategy on Task Performance

Figure 6-8 we give results for methods which learn full plan feasibility models as opposed to action feasibility models. While Complete performs better than Random Plans, it still does not perform as well as the Sequential Actions method. This shows the usefulness of considering both individual action feasibility and information gain when selecting plans to learn from.

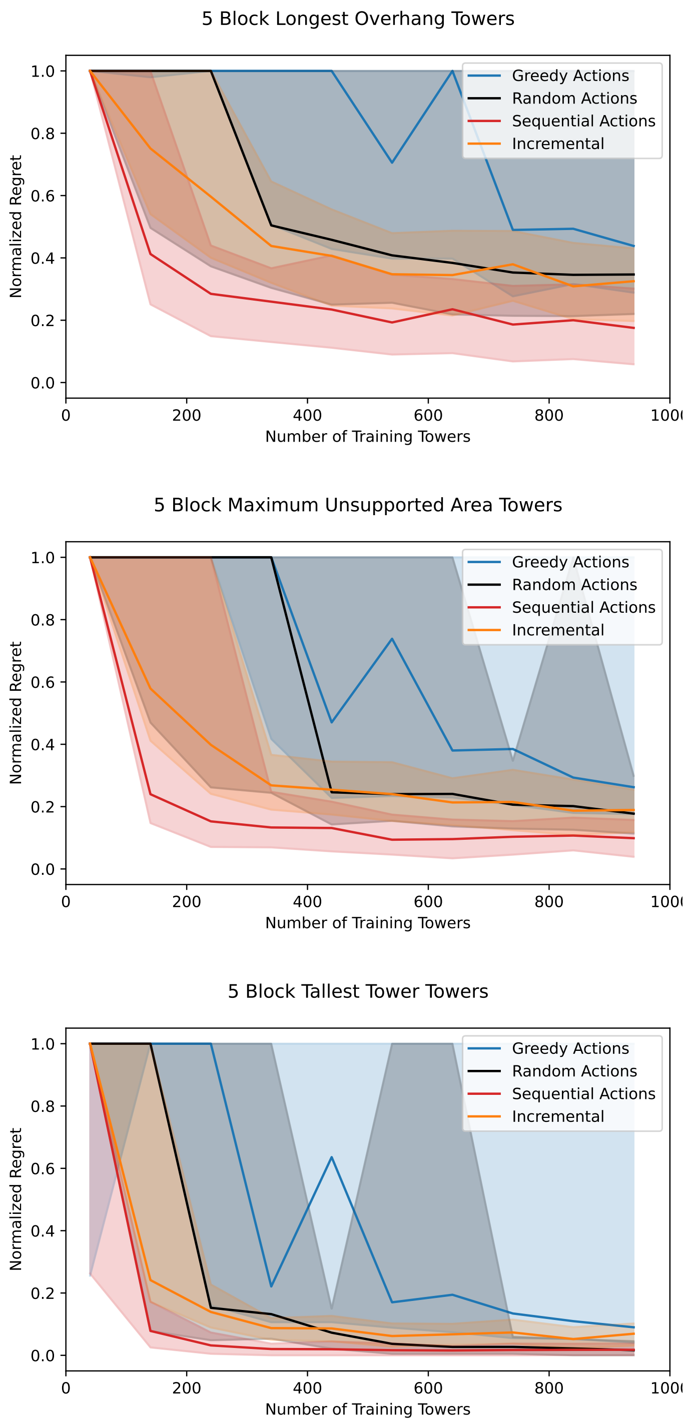


Figure 6-7: **Results for learning an action feasibility model.** A comparison of sampling strategies which learn an AFM on different downstream tasks all performed in simulation. Each method evaluation consists of 4 separate AFM model-learning runs, and each point is the Median Normalized Regret of 50 individual planning runs per learned model. The shaded regions show 25% and 75% quantiles.

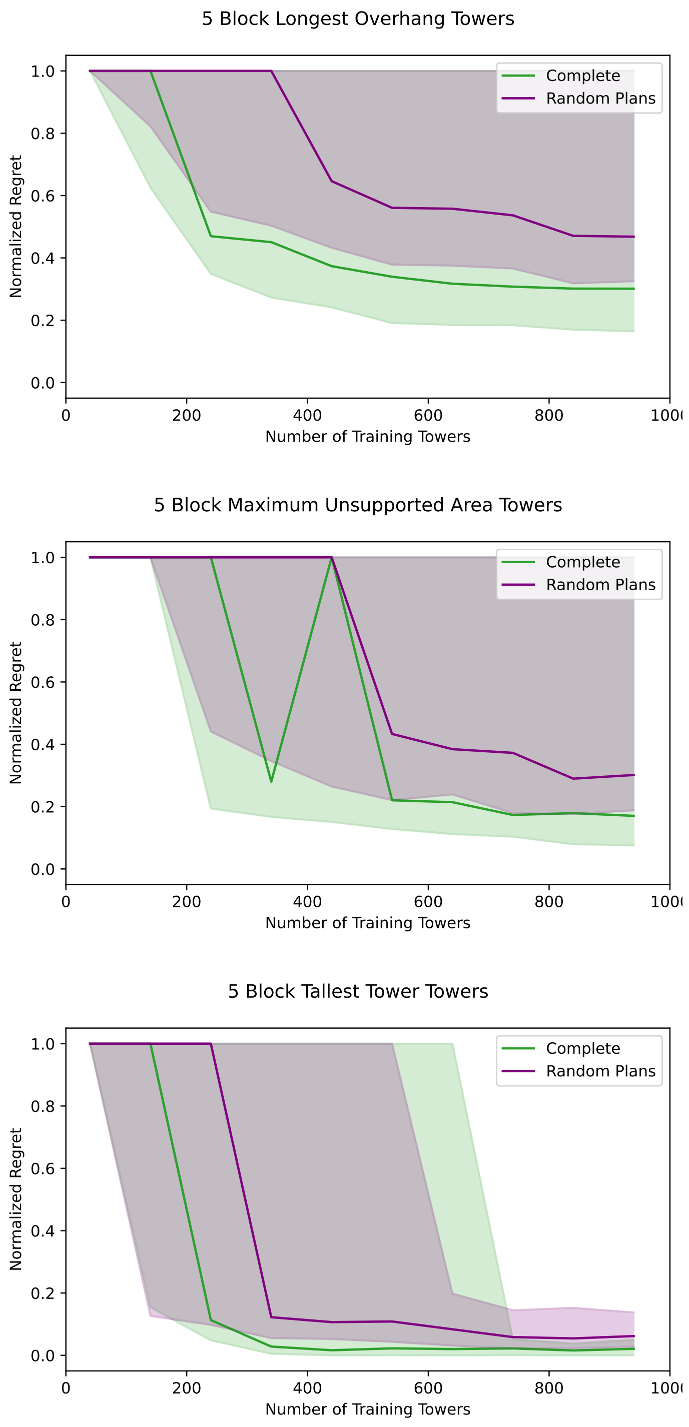


Figure 6-8: **Results for learning a plan feasibility model.** A comparison of sampling strategies which learn plan feasibility models on different downstream tasks all performed in simulation. Each method evaluation consists of 4 separate AFM model-learning runs, and each point is the Median Normalized Regret of 50 individual planning runs per learned model. The shaded regions show 25% and 75% quantiles.

6.4.4 Effect of Model Architecture on Action Feasibility Model Accuracy and Generalizability

We compare our TGN to the other network architectures shown in Figure 6-5. For this evaluation, we report accuracy on a held-out test set of towers built with the novel evaluation blocks. The test set consists of half feasible and half infeasible towers, and 1000 towers for each tower size. Our models were trained in simulation on towers consisting of up to 5 blocks, but our results give model accuracy for towers ranging from 2 to 7 blocks, shown in Figure 6-9.

In the towers domain, it is necessary to consider the joint centers of mass for groups of blocks above support blocks. While the LSTM architecture could remember the previous blocks as it iterates through the tower, in practice we find that it is outperformed by the graph network architectures. We believe this is because the connectivity of the graph networks allows them to precisely compare adjacent blocks in addition to aggregating information about multiple blocks. The weakness of the LSTM is more pronounced as the number of blocks in a tower increases.

Our TGN architecture is structured to be biased towards our particular domain, so it is able to improve its predictions much faster than the other architectures. This enables good planning time performance as seen in Section 6.4.2, with similar long-term performance to the FCGN architecture.

6.4.5 Real Robot Experiments

Finally, we give results for executing the entire active learning pipeline on a real Panda robot. During the learning-phase, the robot built 400 towers over a period of 55 hours using a fixed set of 10 blocks. The TGN ensemble was initialized with 40 random 2-block towers labeled in simulation with added relative-pose noise. Here we look into evaluation-phase task performance when we replace the learned AFM with other possible feasibility models (with a novel set of 10 blocks). Specifically, we compare the following three models:

Analytical We compare to a hand-engineered model of action feasibility that

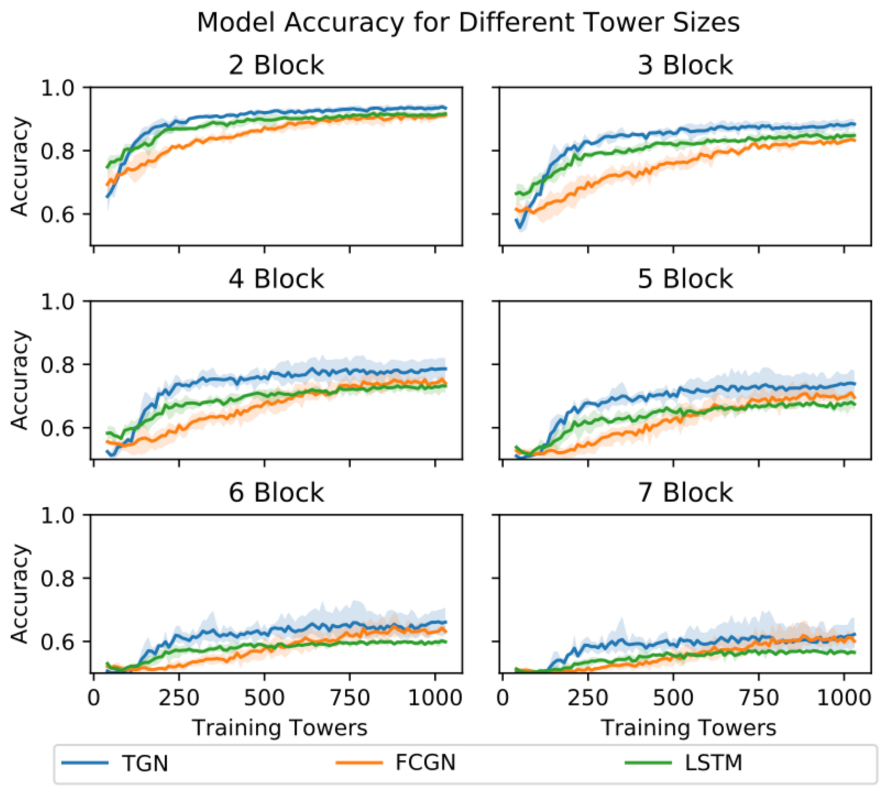


Figure 6-9: **Comparison of NN architectures for action feasibility model learning.** A comparison of different network architectures using the Sequential Actions strategy performed in simulation. The accuracy for each method is averaged over 3 separate training runs. The shading shows the minimum and maximum accuracy from these runs.

calculates whether a candidate block placement is feasible in a noiseless world. The analytical calculation in described in Figure 6-4.

Simulation We also compare to a noisy simulator feasibility model, which predicts that an action is feasible only if the candidate tower is also stable to 10 normally distributed perturbations, with a standard deviation of 5mm, for each block placement.

Learned We perform the Incremental active learning strategy described in Section 6.4.1.

For each task and learned feasibility model we select 5 blocks at random from the evaluation set and plan to maximize the given task reward. The robot constructs 10 towers for each task and feasibility model. In Table 6.1, we report average normalized regret, and the number of total trials that resulted in a stable tower. Figures 6-10, 6-11, and 6-12 includes images of the robot performing the different tasks using these different AFMs.

From these results, it can be seen that the Analytical model can build towers with high reward when the tower is stable, but the towers it chooses to build are rarely stable across all three tasks. However, the Simulation and Learned AFM models can still build towers with large overhang while considering the effects of noisy action execution on a real robot. Our Learned model performs competitively with the Simulation model, and in aggregate leads to similar stability across all tasks (27 versus 24 stable towers out of 30). However, note that the Simulation model

	Tallest Tower		**Longest Overhang**		**Max Unsupported Area**	
Model	**Regret**	**#Stable**	**Regret**	**#Stable**	**Regret**	**#Stable**
Analytical	0.30	7/10	0.80	2/10	0.80	2/10
Simulation	0.41	6/10	0.47	8/10	0.19	10/10
Learned	0.15	9/10	0.45	9/10	0.33	9/10

Table 6.1: **Real robot task performance when using different action feasibility models.** The Learned model was trained with data collected through active learning on the real Panda robot. The Analytical and Simulation models calculate feasibility using the known underlying dynamics but use either no noise or simple noise models respectively.

presents higher variability in tower stability across tasks. This is because the model makes assumptions about the type of noise distribution (Gaussian only in the plane normal to the table) and its parameters (mean and variance), which may be more suitable for certain tasks. The Learned model, on the other hand, may capture other complex real-world phenomena that significantly contribute to action feasibility in a task-agnostic setting.

During the 55-hour learning-phase, the robot encountered 141 errors. Some of which we provisioned for, as discussed in Section 6.3.4, however some required resetting the system by returning all blocks to their home positions manually and moving the robot back to its home position. A block (or more than one) fell off of the table 11 times, requiring us to put the block back on the table for the robot to proceed with planning. There were 29 perception errors which required full resets. This was due to either a block being out of view of the cameras, or estimating the block position with too much error to recover autonomously. Planning failed only 6 times, mostly due to a block falling too close to the robot's base for it to plan a collision-free trajectory to retrieve it. 3 times a block fell so hard that it broke open, requiring reconstruction and a full reset. A block slipped out of the robot's gripper 4 times due to the shifted center of mass or weight of the block, and required a full reset.

6.5 Related Work

A common approach to long-horizon planning problems is to decompose the solution into high-level reasoning over *abstract actions*, and lower level reasoning over *concrete actions* [68, 89, 101]. Singh et al. [93] performed early work on considering the feasibility of high-level plans to improve efficiency when planning in high-dimensional spaces. Recent works have proposed methods that predict feasibility of an action as a way to reduce the number of calls to expensive solvers and enable more efficient planning [22, 104]. Our work builds upon this literature by presenting a method to learn feasibility actively when detailed physical models are not available.

Tasks that involve stacking objects in a *Blocks World* have a long history in artificial intelligence. Early works developed methods to compute the stability of block placements and construct a target configuration [14, 105]. More recently, computer vision researchers have developed scene understanding algorithms that take into account known geometries and stability properties of objects within the scene [37, 47, 87]. Furrer et al. [28] developed a system that can build stacks out of stones using detailed models of the objects.

Recent work has shown the ability to predict tower stability using deep learning techniques [35, 40, 63]. However, typically these works have used passively collected datasets which include orders of magnitude more samples than required in this work — making them infeasible to actively collect on a real robot. An active learning approach allows the robot to explore efficiently, and learn a feasibility model under the real-world noise distribution. In addition, vision-based systems would not be effective when the state contains non-visual properties, such as the center of mass in our stacking domain [35, 63].

6.6 Conclusion

We have presented a system which leverages information-theoretic active learning to acquire an abstract action feasibility model, and shown that incorporating action feasibility into the active learning strategy can dramatically improve sample efficiency. We deployed our system on a real Franka Emika Panda robot arm in a block stacking domain, enabling the robot to learn a useful AFM model with only 400 experiments. In the next chapter we will apply Sequential Actions, as well as an extension, Sequential Goals, to a tool use domain. We believe that this self-supervised method of curious exploration is an exciting direction, as it may someday allow the millions of robots sitting powered-off in laboratories around the world to make effective use of their downtime.

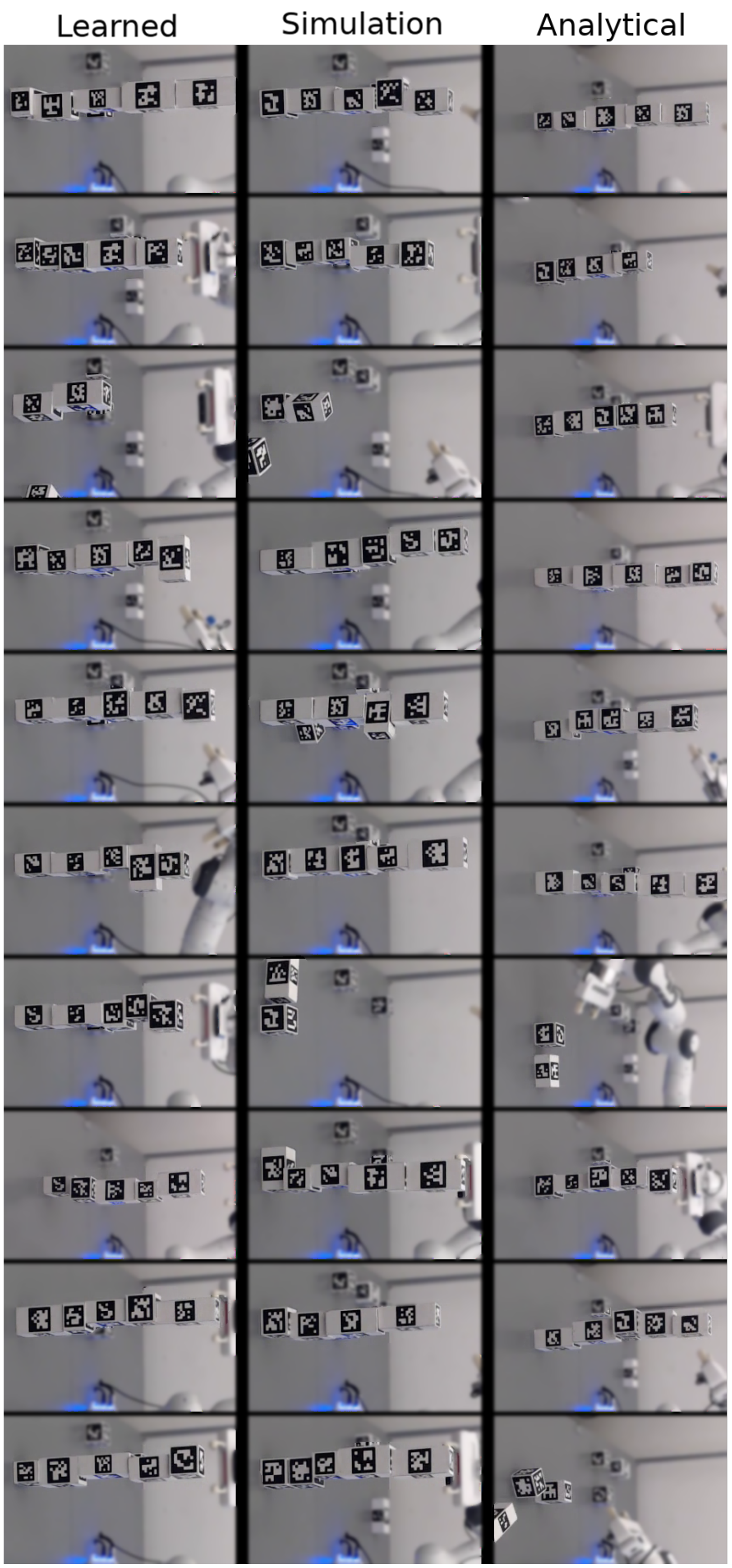


Figure 6-10: **Real robot *Tallest Tower* results.** Towers built for the *Tallest Tower* task when using different AFM models.

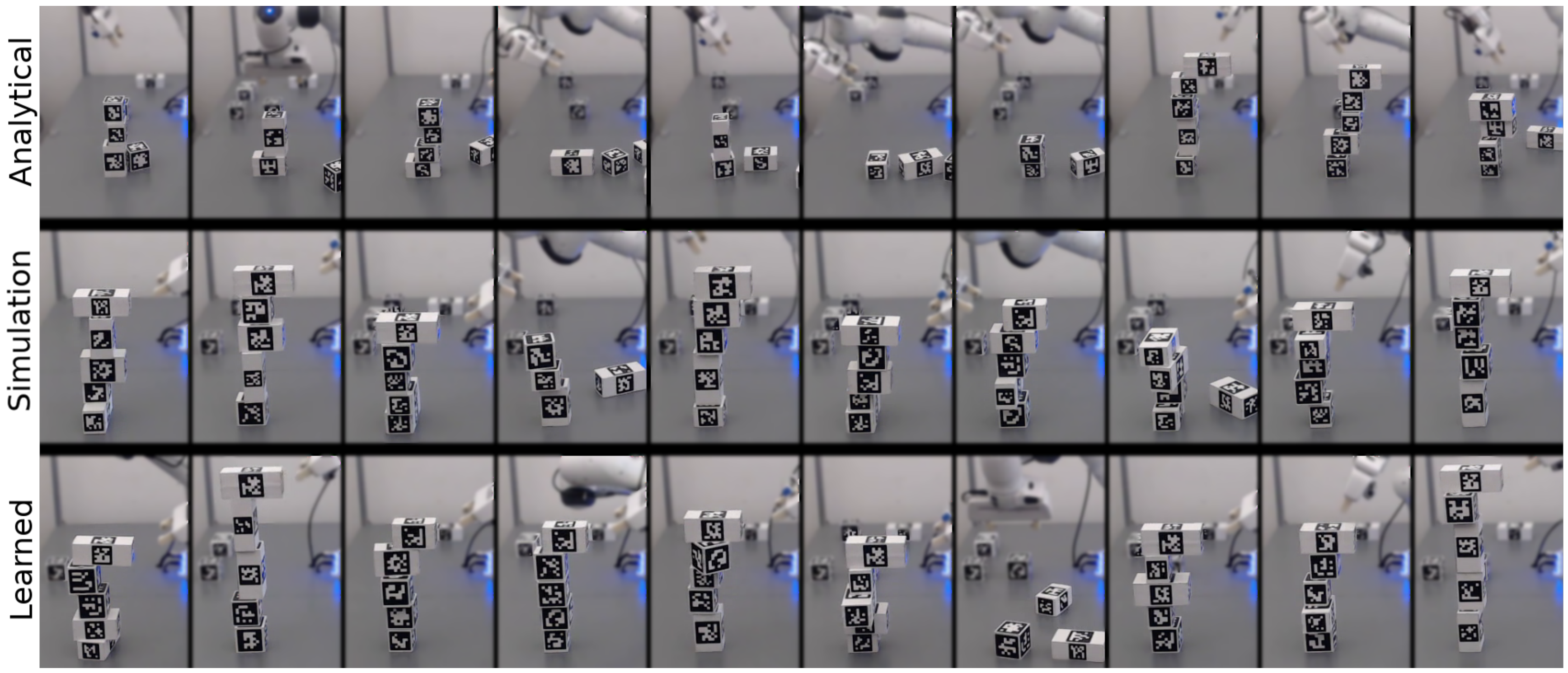


Figure 6-11: **Real robot *Longest Overhang* results.** Towers built for the *Longest Overhang* task when using different AFM models.

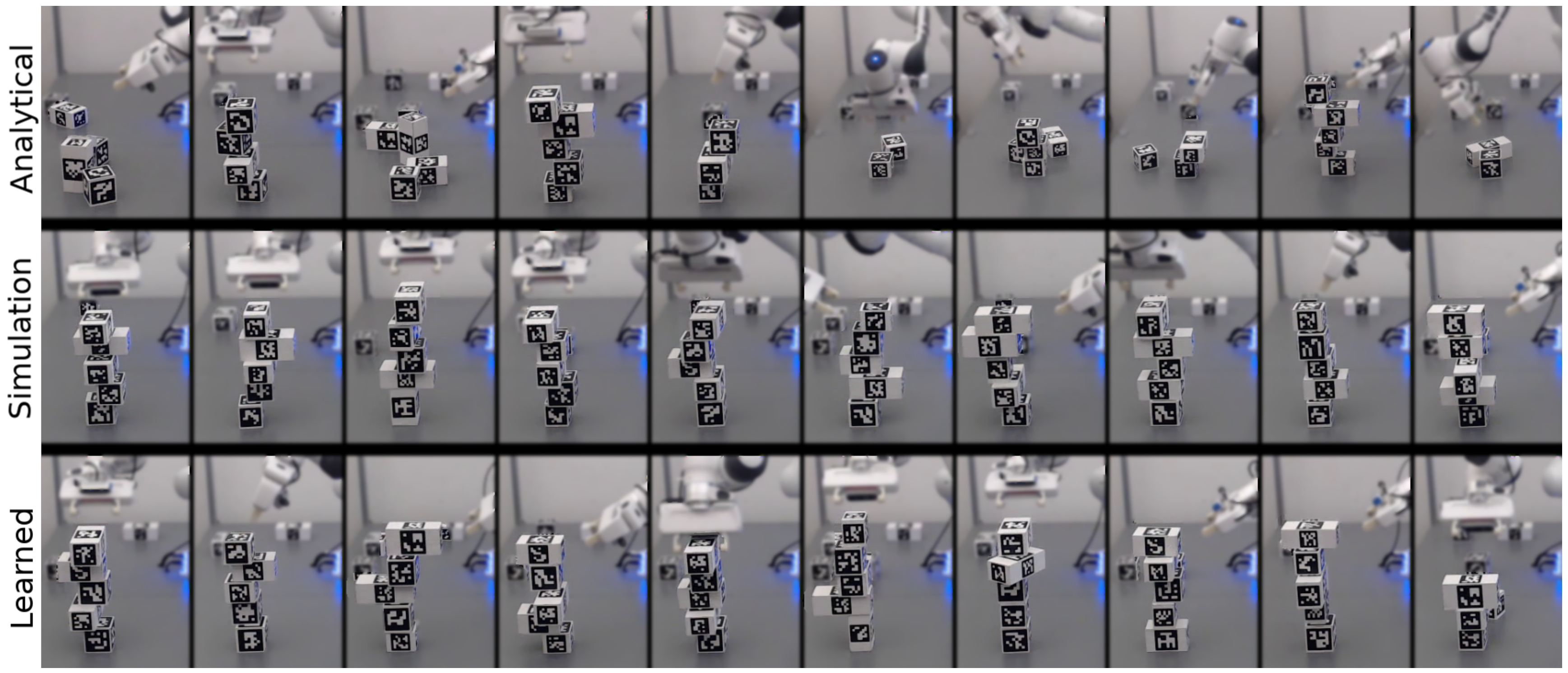


Figure 6-12: **Real robot *Maximum Unsupported Area* results.** Towers built for the *Maximum Unsupported Area* task when using different AFM models.

Chapter 7

Action Feasibility Model Learning with Goals

In this chapter we address limitations of the Sequential Actions strategy for learning action feasibility models (AFMs).[1] In the block stacking domain used in the previous chapter, the action space was reduced to a single abstract stacking action. During active learning, candidate plans were generated by randomly rolling out abstract actions (resulting in P_{act}). This was an effective strategy due to both the number of actions (one single stacking action) and the ability to sample feasible, or stable, plans via random rollouts. Figure 7-1 shows the feasibility labels of several hundred random rollouts of the abstract stacking model. Many of the randomly generated plans are feasible.

During active learning, the robot is attempting to collect samples which enable it to accurately determine the boundary between feasible and infeasible action labels. This requires first acquiring both positive and negative labels so that the robot can then focus on finding the decision boundary. Since it is fairly easy to randomly sample a feasible plan in the block stacking domain, P_{act} was an effective set of candidate plans for determining both the feasible and infeasible subspaces.

[1]The material in this chapter is based on:
Caris Moses, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to Plan with Optimistic Action Models. Workshop on Scaling Robot Learning at IEEE International Conference on Robotics and Automation (ICRA), 2022. [73]

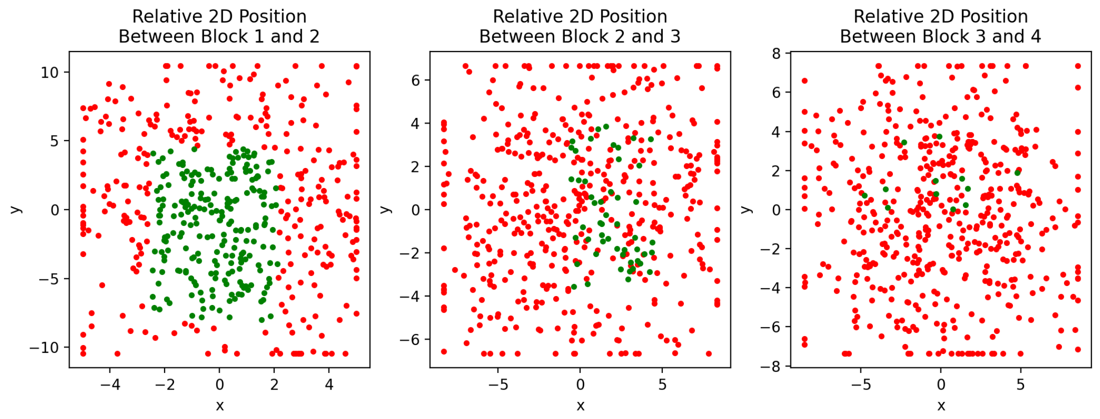


Figure 7-1: **Feasible subspace for block stacking.** These plots show the feasibility labels for 500 randomly generated towers under the optimistic block stacking action model. The optimistic model assumes that any contact between 2 blocks will result in a stable tower. The towers are generated from 4 blocks where each tower has the same order of block placements, but varies the position at which each block is placed in contact with the block below it. The plots show the relative (x, y) position between blocks 1 and 2 (left), 2 and 3 (center) and 3 and 4 (right). The plotted points indicate whether the tower was stable (green) or not (red) after the given block was placed on top. If the tower fell before the given block could be stacked then it is plotted in red. As you can, only sampling 500 plans still results in 12 feasible stable towers.

In many manipulation tasks, however, the action space cannot easily be reduced to a single abstract action. Additionally, the robot may be given optimistic models for which the subspace of feasible actions is very small and thus difficult to randomly sample, as shown in the block pushing task in the introduction (Figure 1-1). In tool use tasks, complex contact and friction constraints lead to very few actions being successful.

In domains with more complex action spaces, more sophisticated methods of generating candidate plans is necessary. In this chapter we evaluate an active learning strategy for learning action feasibility models by generating a rich search space of plans leveraging both optimism and goal-directed planning. Goal-directed planning proves to be a more effective method for generating candidate plans to learn from. We give results in a tool use domain, shown in Figure 7-2, where the robot is tasked with pushing blocks to goal positions, but is only given optimistic models which do not take into account all object and environmental properties.

7.1 Sequential Goals

In many manipulation domains, executing random roll-outs to explore the space of potential plans may not provide enough coverage to elicit interesting training data. Random roll-outs of actions often lead to a robot mostly moving around in free space, and rarely in useful plans such as grasping objects or even making contact with objects. In Chapter 6 we were able to successfully generate candidate plans P_{act} with the help of a useful action space abstraction. However, for problems which cannot be reduced to an abstract action space of a single (e.g. block stacking) action, more sophisticated plan generation methods are required.

As such, during the learning-phase, the Sequential Goals method uses a different plan space, $P = P_{goal}$, to calculate the Sequential objective (Equation 6.1). Now we supply the robot with a space of goal states, $\mathcal{S}_{goal} \subset \mathcal{S}$, which it can sample from and use within a planner to generate a set of candidate plans for active learning

$$P_{goal} = \{a_{1:N} | \exists s_{goal} \in \mathcal{S}_{goal} \text{ s.t. } a_{1:N} = \textsc{plan}(f_O, s_0, s_{goal})\}. \tag{7.1}$$

A set of goal states are sampled, and the optimistic model is used in a planner to generate plans which provide the search space for the optimization of the Sequential objective (Equation 6.1).

As with the Sequential Actions objective, the input to BALD (Equation 4.4) is an action, $x = a$, and the predictions are of individual action feasibility, $y = \phi$.

7.2 Tool Use Domain

We give results in a tool use domain, shown in Figure 7-2 inspired by [108] which was in turn inspired by [100]. The robot is given a tabletop environment with a hook and two blocks. Parameterized actions allow the robot to pick (`Pick`) and place (`Place`) objects, move while holding an object (`MoveHolding`), move while not holding an object (`MoveFree`), and attempt to manipulate the block with the hook tool (`MoveContact`). We define our domain using PDDLStream [33] (described in

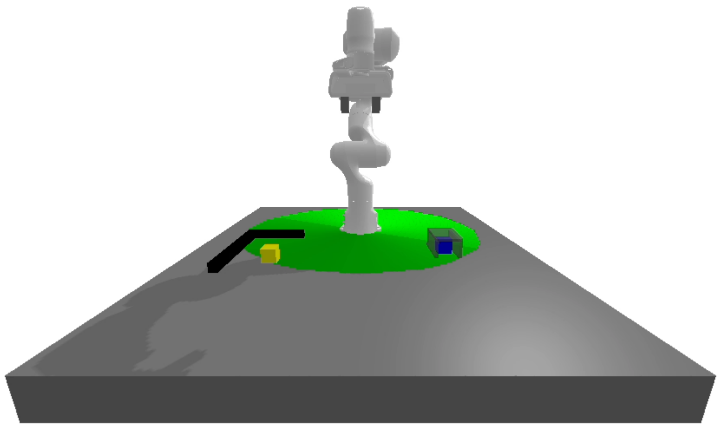


Figure 7-2: **Tool use domain.** The objects have properties which are unknown to the robot. The yellow block is heavy and cannot be picked up or grasped outside of the green circle. The tool must be used to interact with it outside of the green circle. The blue block is inside of an unmodeled tunnel, and the robot must first use the tool to push it out of the tunnel before it can be directly manipulated.

Section 4.3). In the following sections we give the predicates (Section 7.2.1), actions (Section 7.2.2), and goal and initial states (Section 7.2.3) used in our tool use domain.

7.2.1 Predicates

The parameters for our actions and predicates are the following: `?o` is an object, `?p` is an object pose, `?g` is a grasp, `?q` is a robot arm configuration, `?t` is a robot arm trajectory in joint space, `?c` is the relative pose between the tool and an object it is in contact with.

The predicates `AtPose`, `AtConf`, and `HandEmpty` model the changing pose of objects, configuration of the robot arm, and state of the robot gripper. When the robot is holding something, the grasp is modeled with `AtGrasp`. `Block`, `Tool`, and `Table` simply mean that the given object is a block, tool, or table, respectively. `On` indicates that `?o1` is resting stable on top of `?o2`. `PickKin` (`PlaceKin`) indicates that object `?o` can be picked (placed) from (at) `?p` with grasp `?g` following trajectory `?t` from configuration `?q1` to `?q2`.

`FreeMotion` indicates that trajectory `?t` from configuration `?q1` to `?q2`, where the robot is not holding an object, is collision-free. `HoldingMotion` indicates that trajectory `?t` from configuration `?q1` to `?q2`, where the robot is holding object `?o` with grasp `?g`, is collision-free (according to the optimistic model). `ContactMotion` is optimistically modeled, and evaluates to True when planning to move object `?o1` from pose `?p1` to `?p2` using tool `?o2` which is grasped by the robot with grasp `?g` as the robot follows trajectory `?t` from configuration `?q1` to `?q2`. `?c` is the contact configuration between objects `?o1` and `?o2`. This does not take into account the object properties or physical constrains necessary to move two objects together (eg. hooking or poking), and assumes any contact configuration between objects `?o1` and `?o2` will result in object `?o1` moving to pose `?p2`. This assumption is shown in Figure 7-3.

7.2.2 Action Descriptions

The following action descriptions define the optimistic transition model, f_O. The pick and place actions are defined as follows:

```
(:action Pick
  :param (?o1 ?p1 ?o2 ?g ?q1 ?q2 ?t)
  :pre (and (PickKin ?o1 ?p1 ?g ?q1 ?q2 ?t)
            (AtPose ?o1 ?p1)
            (AtConf ?q1)
            (On ?o1 ?o2)
            (HandEmpty))
  :effect (and (AtGrasp ?o1 ?g)
               (AtConf ?q2)
               (not (AtConf ?q1))
               (not (AtPose ?o1 ?p1))
               (not (HandEmpty))
               (not (On ?o1 ?o2)))
(:action Place
  :param (?o1 ?p1 ?o2 ?g ?q1 ?q2 ?t)
```

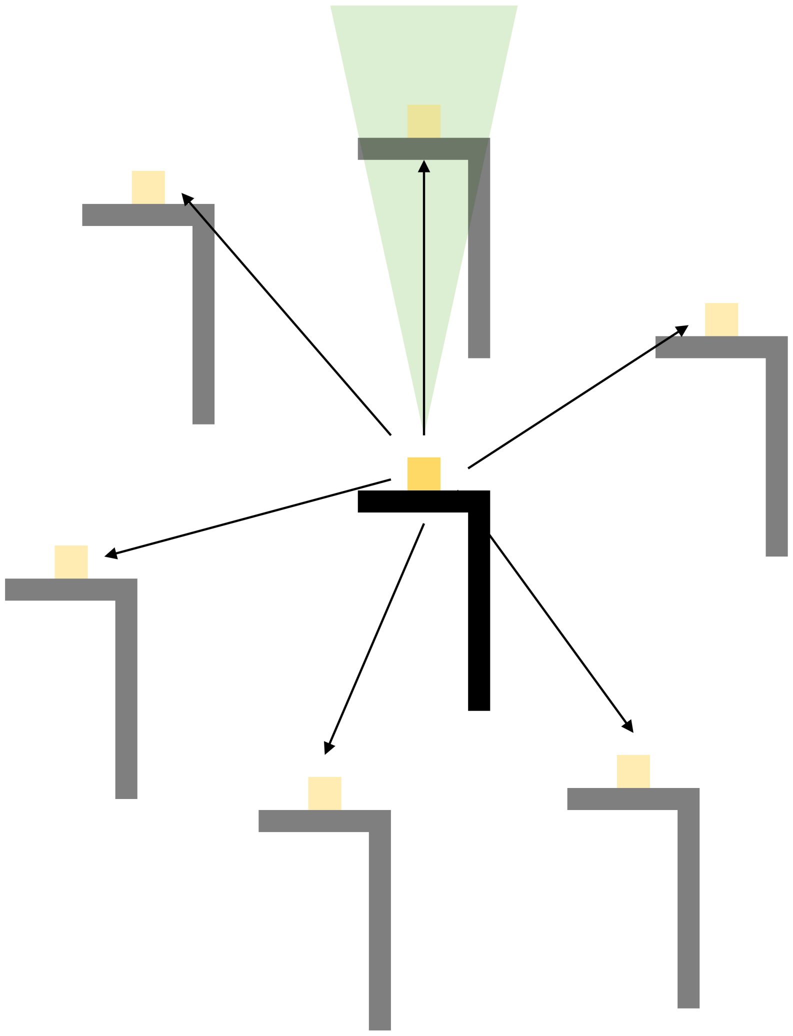


Figure 7-3: **Optimistic pushing model.** The `ContactMotion` predicate evaluates to True for all potential push actions from the initial contact configuration, shown in the center, to all surrounding goal positions. The motion follows a straight line path from the initial to goal pose of the block. This ignores the friction conditions which enable successful pushing. The green region shows roughly where the yellow block could be successfully pushed.

```
  :pre (and (PlaceKin ?o1 ?p1 ?g ?q1 ?q2 ?t)
            (AtGrasp ?o1 ?g)
            (AtConf ?q1))
  :effect (and (AtPose ?o1 ?p1)
               (HandEmpty)
               (AtConf ?q2)
               (On ?o1 ?o2)
               (not (AtConf ?q1))
               (not (AtGrasp ?o1 ?g)))
```

The different types are move actions are defined as follows:

```
(:action MoveFree
  :param (?q1 ?q2 ?t)
  :pre (and (AtConf ?q1)
            (FreeMotion ?q1 ?q2 ?t)
            (HandEmpty))
  :effect (and (AtConf ?q2)
               (not (AtConf ?q1)))


(:action MoveHolding
  :param (?o ?g ?q1 ?q2 ?t)
  :pre (and (HoldingMotion ?o ?g ?q1 ?q2 ?t)
            (AtConf ?q1)
            (AtGrasp ?o ?g))
  :effect (and (AtConf ?q2)
               (not (AtConf ?q1)))
(:action MoveContact
  :param (?o1 ?c ?o2 ?p1 ?p2 ?g ?q1 ?q2 ?t)
  :pre (and (ContactMotion ?o1 ?c ?o2 ?p1 ?p2 ?g ?q1 ?q2 ?t)
            (AtConf ?q1)
            (AtPose ?o1 ?p1)
            (AtGrasp ?o2 ?g)
            (Block ?o1))
  :effect (and (AtConf ?q2)
               (AtPose ?o1 ?p2)
               (not (AtConf ?q1))
               (not (AtPose ?o1 ?p1)))
```

The `MoveContact` action is optimistic due to its `ContactMotion` precondition, in that it ignores the friction cone constraints between the tool and target object which enable successful pushing. The `MoveHolding` action is optimistically modeled because it assumes that once an object is held, it can be moved anywhere. While this model includes kinematic constraints, it does not account for object properties which may make moving while holding an object infeasible (e.g. the weight of the yellow block). Although not demonstrated in our particular domain, the grasp of the in-hand object and the trajectory followed could also impact the success of this action. Finally, the `Pick` action is optimistically modeled because it assumes that an object can be picked from anywhere which is kinematically within reach. This fails when the robot attempts to pick up the yellow block outside of the green circle, and when it attempts to pick the blue block from its initial position inside of the unmodeled tunnel. We learn action feasibility models $\hat{f}_{contact}$, $\hat{f}_{holding}$, and $\hat{f}_{pick}$ for these optimistically modeled actions.

7.2.3 Initial and Goal States

The goal space we give the robot to sample goals from during active learning, $\mathcal{S}_{goal}$, is `(AtPose ?o ?p)` where `?o` can be either the yellow block or the blue block, and `?p` is a pose on the table. For each planning and execution instance the initial state is the same. The yellow block begins in the same position within the green circle, and the blue block begins in the tunnel. However the goals states are randomly generated during learning and evaluation.

The planner is given the following goal predicate

```
(AtPose GoalObj GoalPose)
```

and initial state

```
((AtPose GoalObj GoalObjInitPose) ∧
(AtPose Tool ToolPose) ∧
(Block YellowBlock) ∧
(Block BlueBlock) ∧
(Tool Tool) ∧
(Table Table)).
```

`GoalObj` can be either `YellowBlock` or `BlueBlock`.

7.3 Implementation

All experiments were conducted in PyBullet [18]. We discuss the details of the planner in Section 7.3.1, and the representation of the learned feasibility models in Section 7.3.2.

7.3.1 Planning

In many planning problems there are potentially multiple solutions, or plans, which can achieve a given goal. Most modern planners, including PDDLStream [33] (described in Section 4.3.1), are only able to return a single or very few solutions when given a specific planning problem. Our Sequential Goals method relies on plans returned from a planner to generate plan space P_{goal}. If we generate plans using PDDLStream, then for a given goal, $s_{goal} \in \mathcal{S}_{goal}$, the robot will only ever sample (and execute) a very small subset of the plans which can possibly achieve s_{goal}. Additionally, P_{goal} is generated using the optimistic model. Therefore, not only will the plans not sufficiently cover the space of plans which could possibly achieve s_{goal}, they also might not be feasible to execute under the unknown true dynamics.

Take, for example, a goal of getting the blue block to a specific pose on the table which is outside of the tunnel. Under the optimistic model, the robot believes that it can directly pick up the blue block from its initial position and move it to the goal position. However, under the true transition model, the tunnel is blocking the robot from doing so. If PDDLStream only ever returns this plan, of directly picking and placing the blue block, then it will never discover the alternative plan (also a valid solution under the optimistic model) of first using the tool to push the blue block out of the tunnel, then directly picking and placing the block at the goal position.

Therefore, we use *skeleton-based planning* to implement PLAN when generating P_{goal} (Equation 7.1) and achieve diverse plans under the optimistic model. Our skeleton-based planner takes in an optimistic transition model defined by PDDLStream-style action descriptions, an initial state state and a goal state both represented by logical predicates, and a plan skeleton. A plan skeleton is a sequence of actions where some of the action parameters are not yet grounded, meaning they do not have a specific value. Ungrounded variables begin with # and are grounded at planning time. In the tool use domain, the following plan skeletons are used to generate plans to achieve goal states sampled from $\mathcal{S}_{goal}$:

1.
   ```
   MoveFree -> Pick(Tool, ToolPose, Table, #g) ->
   MoveHolding(Tool, #g) ->
   MoveContact(Tool, #c, GoalObj, GoalObjInitPose, GoalPose, #g)
   ```

2.
   ```
   MoveFree ->
   Pick(GoalObj, GoalObjInitPose, Table, #g) ->
   MoveHolding(GoalObj, #g) ->
   Place(GoalObj, GoalPose, Table, #g)
   ```

3.
   ```
   MoveFree -> Pick(Tool, ToolPose, Table, #g1) ->
   MoveHolding(Tool, #g1) ->
   MoveContact(Tool, #c, GoalObj, GoalObjInitPose, #p1, #g1) ->
   MoveHolding(Tool, #g1) ->
   Place(Tool, #p2, Table, #g1) ->
   MoveFree ->
   Pick(GoalObj, #p1, Table, #g2) ->
   MoveHolding(GoalObj, #g2) ->
   Place(GoalObj, GoalPose, Table, #g2)
   ```

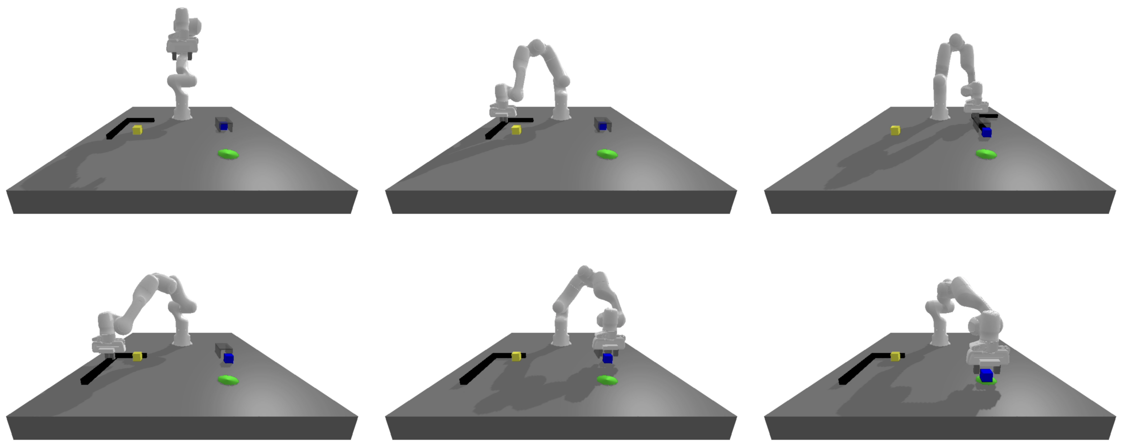


Figure 7-4: **Simulated Panda executing a grounded plan skeleton.** A time-lapse of the robot executing a grounded plan from Skeleton #3 for the blue block. Note that this plan is feasible under the true transition function, however when planning with the optimistic model it is possible (and far more likely) to generate a plan which is infeasible under the true transition model.

We omit the configuration and trajectory parameters of each action for readability, but these parameters are also grounded at planning time.

The first skeleton has the robot use the tool to push `GoalObj` to the `GoalPose`. The second has the robot directly pick `GoalObj` and place it at the `GoalPose`. The third skeleton has the robot first use the tool to push `GoalObj`, then place the tool, then pick `GoalObj` and place it at `GoalPose` A visual of a grounded (feasible) plan generated from Skeleton #3 is given in Figure 7-4. While we do not directly use PDDLStream [33] for planning, many of its tools were useful in developing our skeleton-based planner.

7.3.2 Learning

In order to learn efficiently, we use a low-dimensional state and action representation as input to the learned feasibility models. The input to $\hat{f}_{contact}$ is the (x, y) position of the intended final push pose of the pushed block. The robot receives label $\phi = 1$ if the block is successfully pushed to the final pose, and $\phi = 0$ otherwise. The input to $\hat{f}_{holding}$ is the (x, y) position of the intended final pose of the held block within the robot's gripper. The robot receives label $\phi = 1$ if the block successfully reaches the final pose, and $\phi = 0$ otherwise. The input to $\hat{f}_{pick}$ is the (x, y) position of the picked block's initial pose. The robot receives label $\phi = 1$ if the block is successfully picked from the table, and $\phi = 0$ otherwise. Each PDDL action has a corresponding low-level controller which takes a grounded action and returns the corresponding trajectories and grasps. In our implementation these actions do not alter the orientations of the blocks. However, if we added rotations to the robot's abilities, we would then augment these learned AFMs to also take in the rotation of the block.

We learn a separate model for each of the blocks. In addition, the contact configuration between the pushed block and the tool, as well as the grasp of the tool in hand can impact the success of the `MoveContact` action. As such we learn a separate model for every combination of block, contact configuration, and grasp for the `MoveContact` actions, and just a single model for each block's `Pick` and `MoveHolding` action.

The feasibility models are each represented by an ensemble of Multi-Layer Perceptions (MLPs). We found that the distribution used to sample initial NN weight values and the type of activation function in the MLPs greatly impacted the quality of the ensemble's predictions. Initially, we used ReLU (Rectified Linear Unit) activation functions between each linear layer, and the default PyTorch [84] weight initialization method. In PyTorch, by default, weights are drawn from the distribution Uniform($\frac{-1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}$), where n_{in} is the dimensionality of the input to the linear layer. In our MLPs the value of n_{in} is either 2 to 48 depending on whether or not the layer is the initial input to the MLP. While this resulted in models which were able to accurately predict feasibility, the ensemble's predictions had high confidence in areas with no training data. This phenomenon is visualized in Figure 7-5. In order to accurately represent the distribution $\hat{f}_A(s, a; \Theta)$, we desire a model which makes confident predictions near data and is uncertain far from data. This allows the robot to consider uncertainty when selecting actions to execute.

We found that widening this distribution to Uniform$(-1, 1)$ resulted in initial models with more diverse predictions which enabled them to converge to very different, but accurate, models. This diversity resulted in improved predictions in areas with little data. However, we still found uncertainty to be low *between* datapoints with the same label. This is also shown in Figure 7-5.

We then explored using a Sigmoid activation function as opposed to a ReLU, and changing the distribution which weights are initially drawn from to a Normal$(0, \sigma)$ distribution. For certain values of σ this enabled us to get the ensemble to make predictions with high uncertainty far from data *and between* datapoints of the same label, which is desired. The value of σ impacts how tightly the high confidence area around datapoints hugs the samples, as is shown in Figure 7-6. In this way it acts as a tuneable parameter for determining how samples should impact nearby predictions.

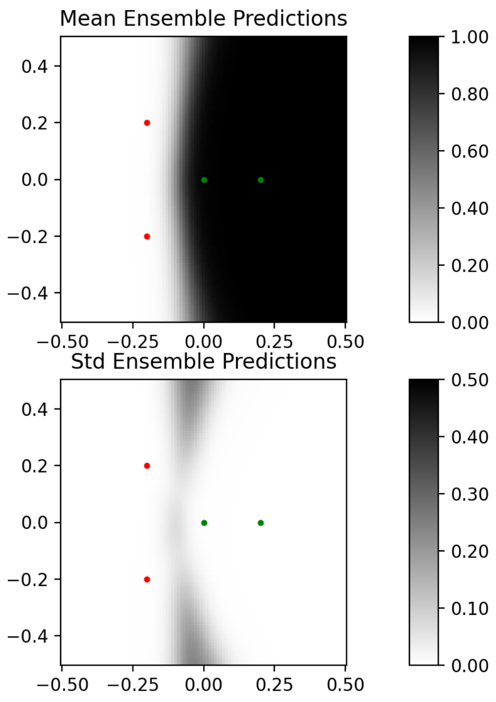
We found an ensemble of 20 MLPs to be effective at modeling a distribution over the predictive space. Using a wider distribution to sample initial weight values is beneficial for getting diverse models, however it also makes it more challenging for models to converge to accurate predictions as initial weight values are potentially very far from 0. To deal with this, we perform up to 5 restarts if the models have not converged to a training loss of less than ϵ after 300 epochs. We used $\epsilon = 1$ in our experiments. We also found that learning improved if instead of selecting the plan in P_{goal} which maximizes the Sequential objective (Equation 6.1), we randomly sample from the top 1% of plans in P_{goal}.

7.4 Evaluation

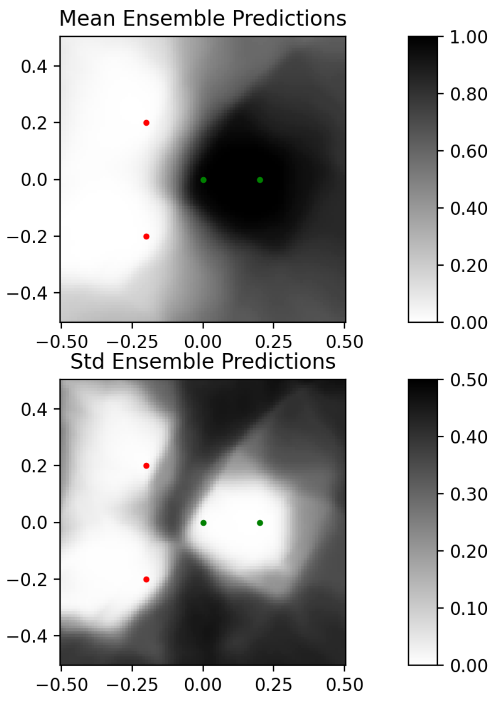
First we discuss the baselines we compare against in Section 7.4.1. Then we evaluate the learning efficiency of the proposed Sequential Goals method over the previous Sequential Actions method and compare to other baselines in Section 7.4.2. Finally, we give results for using the learned actions within a planner in Section 7.4.3.

7.4.1 Baselines

We compare the Sequential methods, Sequential Goals and Sequential Actions, to random baselines. Random Goals randomly samples from P_{goal} and Random Actions randomly samples from P_{act}, where P_{act} consists of plans generated from random

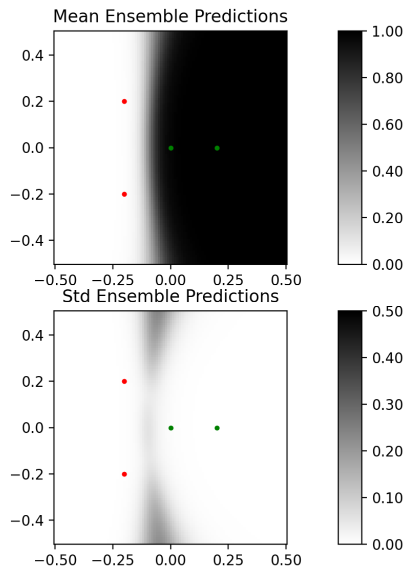


(a) $w_0 \sim \text{Uniform}(\frac{-1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}), n_{in} \in \{2, 48\}$

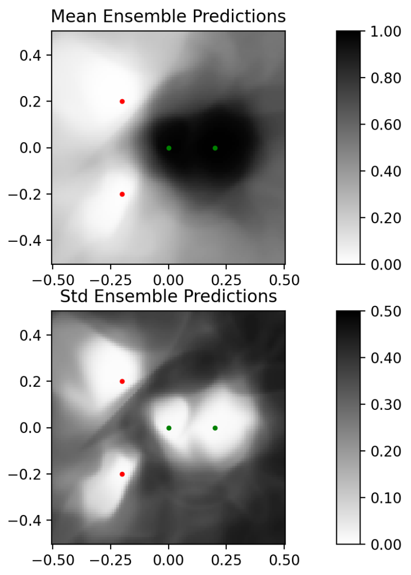


(b) $w_0 \sim \text{Uniform}(-1, 1)$

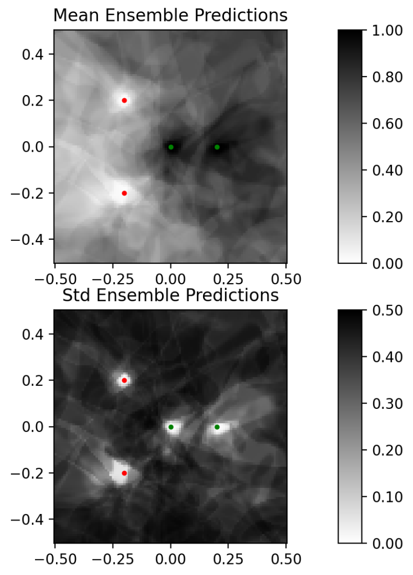
Figure 7-5: **ReLU with initial weights drawn from a uniform distribution.** These plots show the mean (top) and standard deviation (bottom) predictions from an ensemble of 20 MLP. Each MLP takes in a 2D input and make predictions $\in [0, 1]$. The points indicate positive (green, label is 1) and negative (red, label is 0) samples in the dataset. These images show predictions from learned models with initial weights, w_0, drawn from a Uniform distribution. The left is the default PyTorch weight initialization. On the right we show how widening the uniform distribution enables more accurate uncertainty predictions far from datapoints. However, we still desire high uncertainty *between* datapoints.



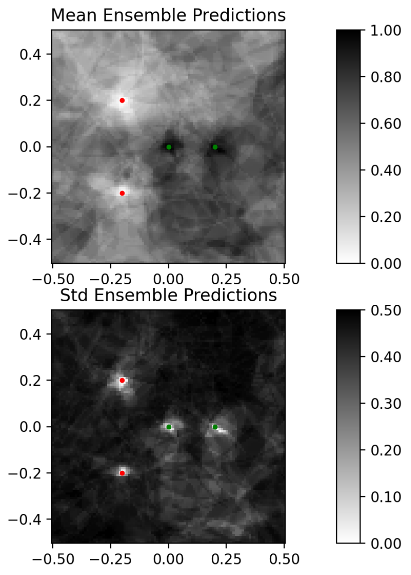
(a) w_0 ∼Normal$(0, 1)$



(b) w_0 ∼Normal$(0, 3)$



(c) w_0 ∼Normal$(0, 5)$



(d) w_0 ∼Normal$(0, 7)$

Figure 7-6: **Sigmoid with initial weights drawn from a normal distribution.** These plots show the mean (top plots) and standard deviation (bottom plots) predictions from an ensemble of 20 MLP. Each MLP takes in a 2D input and make predictions $\in [0, 1]$. The points indicate positive (green, label is 1) and negative (red, label is 0) samples in the dataset.

Figure 7-6: (cont.) These images show predictions from learned models with initial weights, w_0, drawn from a Normal$(0, \sigma)$ distribution. The value of σ changes uncertainty predictions around datapoints. The higher the value, the tighter the low uncertainty region hugs sampled points.

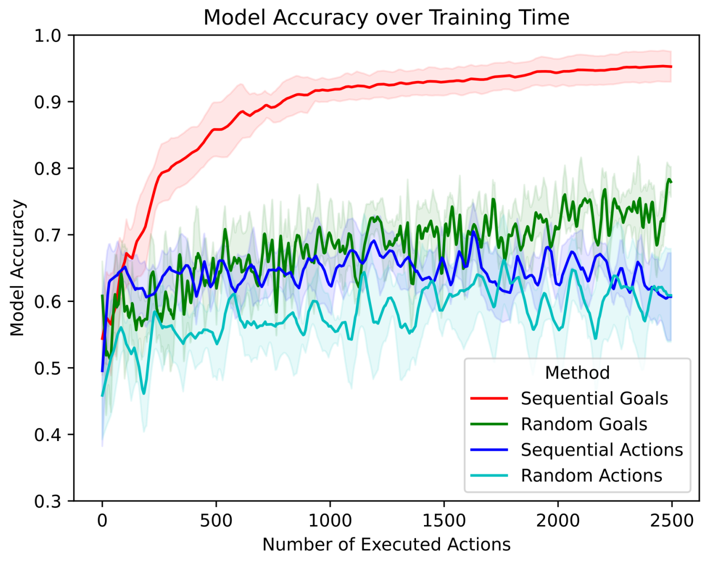


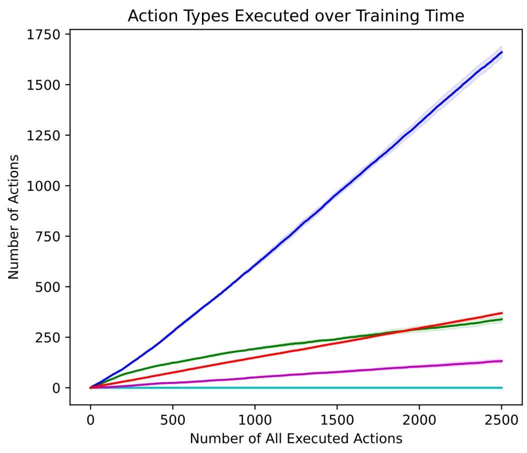
Figure 7-7: **Joint model accuracy for all learned action feasibility models.** Each line consists of 5 runs. The Sequential methods use 20 MLPs in their ensembles, and the random methods (which do not perform active learning) use a single MLP to represent each $\hat{f}_A$. The x-axis is all actions executed by the robot including ones which were not modeled optimistically (e.g. place).

roll-outs of the optimistic transition model,

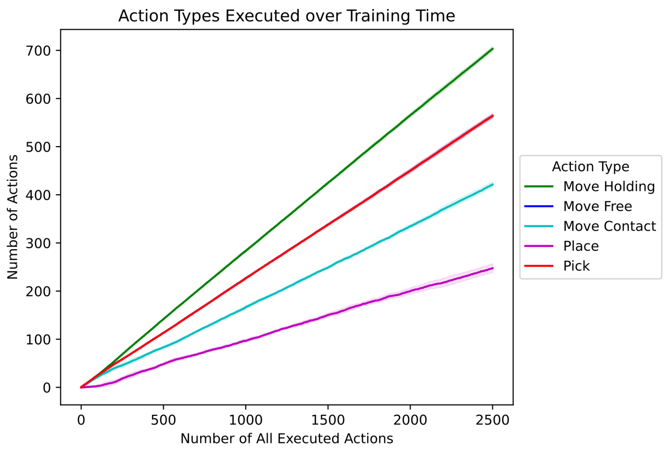
$$P_{act} = \{a_{1:N} | \exists s_{0:N} \text{ s.t. } s_i = f_O(s_{i-1}, a_i)\}.$$

7.4.2 Learning Efficiency

Figure 7-7 shows the accuracy of the learned feasibility models for all methods. Sequential Goals is able to quickly learn the true underlying feasibility of the optimistic models. The performance of both Sequential Goals and Random Goals shows the usefulness of $\mathcal{S}_{goal}$ and using goals to generate the optimization search space. The benefits of using the Sequential objective (Equation 6.1) to actively acquire trajectories is evidenced by the jump in performance between Sequential Goals and Random



(a) Sequential Actions

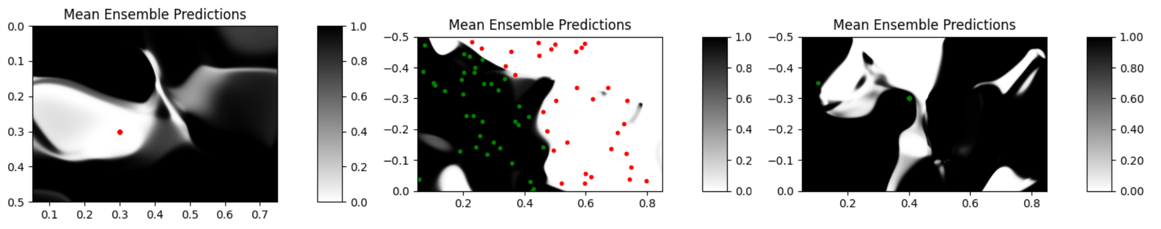


(b) Sequential Goals

Figure 7-8: **Actions selected by different Sequential strategies.** The types of actions selected by the Sequential Actions strategy are shown on the left, and the types of actions selected by the Sequential Goals strategy are shown on the right. These are averaged over 5 runs and standard deviation is also visualized. In the Sequential Goals plot, the `MoveFree` and `Pick` actions perfectly correspond. Each time the robot picks up an object it must first move through free space.

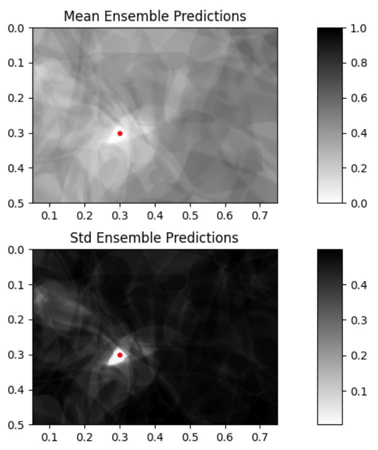
Goals which randomly samples from P_{goal}. Sequential Actions and Random Actions perform poorly because the plans in P_{act} are not goal-directed and thus consist of many actions which are not useful, such as moving without any objects in hand. Figure 7-8 shows the different types of actions executed by each sequential method. As you can see the Sequential Actions strategy ends up executing mostly `MoveFree` actions which do not aid in learning the feasibility models.

Figures 7-9, 7-10, 7-11, and 7-12 show both the mean and standard deviation of the AFM ensemble predictions for each method after a single learning-phase run of 2500 executed actions. They also visualize the samples selected and executed by the robot. You can see in the figures that the exploration strategy greatly impacts the quality of the collected data and the resulting learned AFM. We only show figures for optimistic actions which the robot attempted to execute during the learning-phase. As such, the number of plots for each method varies. Only the Sequential Goals method has the robot attempt all optimistic actions (`Pick`, `MoveHolding`, and `MoveContact`) for both the yellow and blue block.

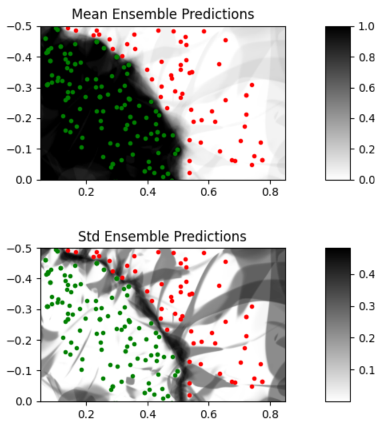


(a) Blue Block $\hat{f}_{pick}$ (b) Yellow Block $\hat{f}_{holding}$ (c) Yellow Block $\hat{f}_{pick}$

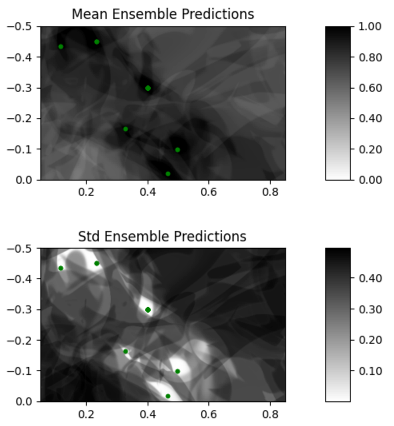
Figure 7-9: **Learned action feasibility model predictions for Random Actions.** These plots show the ensemble predictions of the AFMs learned from a single run of the Random Actions strategy. The grayscale color indicates the model's prediction for the given (x, y) input (see Section 7.3.2 for the model inputs). A value of 1.0 indicates feasible action parameters, and 0.0 indicates infeasible action parameters. The random methods do not require maintaining a distribution over the parameter space, so we only use an ensemble consisting of a single model. The plotted points are the action parameters which were executed by the learning strategy and were found to be either feasible (green) or infeasible (red). Only the actions that were used to train the AFMs are visualized. Many other actions (such as `MoveFree`) were executed, but not used to learn an AFM and thus are not visualized here. For the blue block, the robot attempts to pick it from its initial position, but finds that that action is infeasible. It never explores plans to first push the blue block out of the tunnel, therefore no other pick actions are ever successful. For the yellow block the robot is able to successfully pick it up from its initial position. It is then able to learn that some `MoveHolding` actions are feasible while others are not. The Random Actions strategy leads the robot to execute a plan which involve picking, placing, then picking up the yellow block again. This is shown by the `Pick` attempt which does not correspond to the yellow block's initial position. No `MoveContact` actions are ever attempted due to the difficulty of randomly sampling a successful tool use plan.



(a) Blue Block $\hat{f}_{pick}$

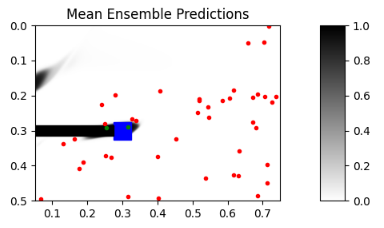


(b) Yellow Block $\hat{f}_{holding}$

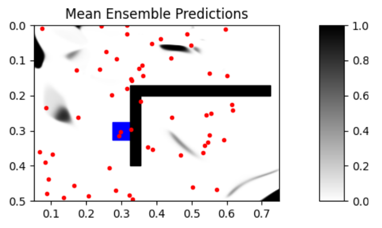


(c) Yellow Block $\hat{f}_{pick}$

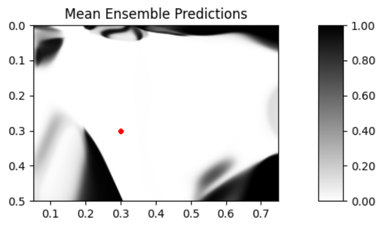
Figure 7-10: **Learned action feasibility model predictions for Sequential Actions.** These plots show the ensemble predictions of the AFMs learned from a single run of the Sequential Actions strategy. The grayscale color indicates the model's prediction for the given (x, y) input (see Section 7.3.2 for the model inputs). In the top mean prediction plots, a value of 1.0 indicates feasible action parameters, and 0.0 indicates infeasible action parameters. In the bottom standard deviation plots the value indicates the standard deviation of the ensemble's prediction for a given AFM input. The plotted points are the action parameters which were executed by the learning strategy and were found to be either feasible (green) or infeasible (red). Only the actions that were used to train the AFMs are visualized. Many other actions (such as `MoveFree`) were executed, but not used to learn an AFM and thus are not visualized here. For the blue block, the robot attempts to pick it from its initial position, but finds that that action is infeasible. It never explores plans to first push the blue block out of the tunnel, therefore no other pick actions are ever successful. For the yellow block the robot is able to successfully pick it up from its initial position. It is then able to learn that some `MoveHolding` actions are feasible while others are not. The Sequential Actions strategy leads the robot to execute a plan which involve picking, placing, then picking up the yellow block again. This is shown by the `Pick` attempts which do not correspond to the yellow block's initial position. No `MoveContact` actions are ever attempted due to the difficulty of randomly sampling a successful tool use plan when generating P_{act}.



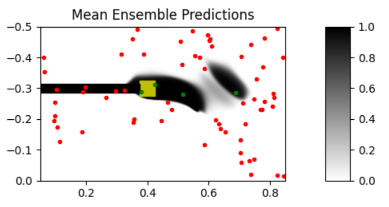
(a) Blue Block $\hat{f}_{contact}$



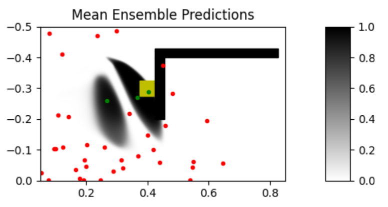
(b) Blue Block $\hat{f}_{contact}$



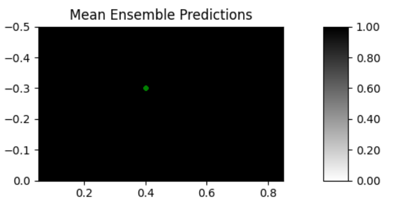
(c) Blue Block $\hat{f}_{pick}$



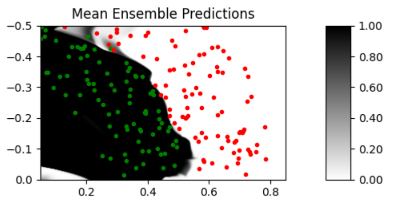
(d) Yellow Block $\hat{f}_{contact}$



(e) Yellow Block $\hat{f}_{contact}$



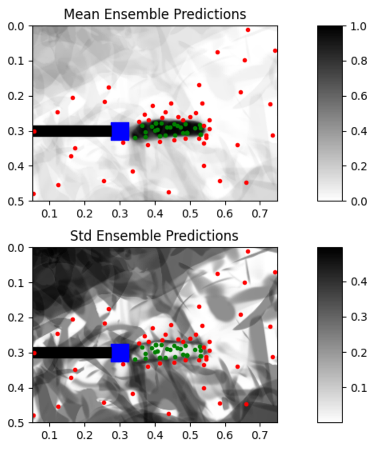
(f) Yellow Block $\hat{f}_{pick}$



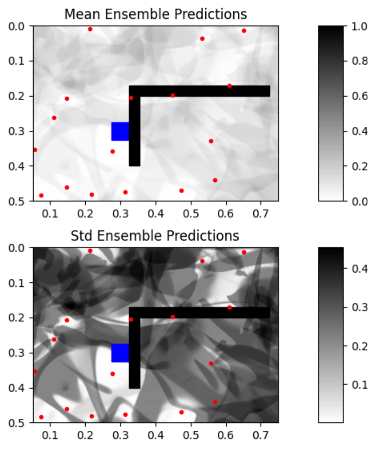
(g) Yellow Block $\hat{f}_{holding}$

Figure 7-11: **Learned action feasibility model predictions for Random Goals.** These plots show the ensemble predictions of the AFMs learned from a single run of the Random Goals strategy. The grayscale color indicates the model's prediction for the given (x, y) input (see Section 7.3.2 for the model inputs). A value of 1.0 indicates feasible action parameters, and 0.0 indicates infeasible action parameters. The random methods do not require maintaining a distribution over the parameter space, so we only use an ensemble consisting of a single model. The plotted points are the action parameters which were executed by the learning strategy and were found to be either feasible (green) or infeasible (red). For the $\hat{f}_{contact}$ AFMs we also visualize the contact configuration between the tool, shown in black, and the block, shown in its color in its initial position. From this contact configuration the robot attempts to move it to the different (x, y) positions in the plot. For the blue block $\hat{f}_{contact}$ AFM shown in (a), the robot receives a few positive labels near the blue block's initial position, but is never able to randomly sample a plan which successfully pushes the blue block out of the tunnel (which extends to $x = 0.4$).

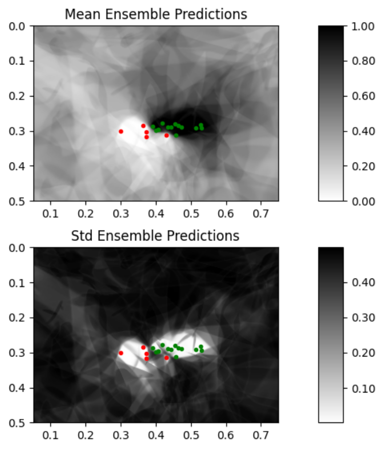
Figure 7-11: (cont.) The $\hat{f}_{contact}$ AFM shown in (b) is never feasible due to the tool colliding with the tunnel before it is ever able to reach the desired contact configuration. The robot attempts to pick the blue block from its initial position, but finds that that action is infeasible due to the tunnel. Since it is never able to successfully pick the blue block, no `MoveHolding` actions are ever attempted with the blue block. For the yellow block the robot is able to see some positive labels for both $\hat{f}_{contact}$ AFMs shown in (d) and (e). It is also able to successfully pick the yellow block from its initial position. Finally it executes many `MoveHolding` actions to learn $\hat{f}_{holding}$ for the yellow block.



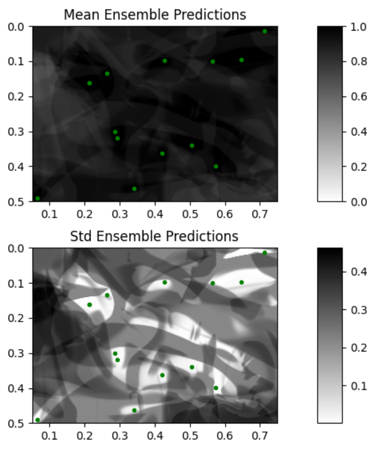
(a) Blue Block $\hat{f}_{contact}$



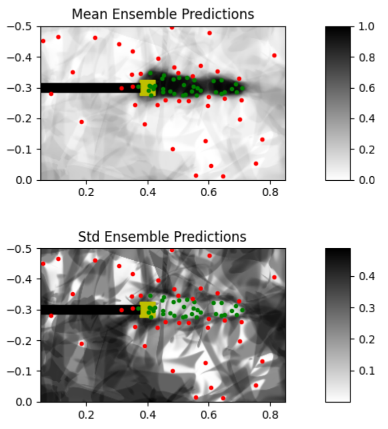
(b) Blue Block $\hat{f}_{contact}$



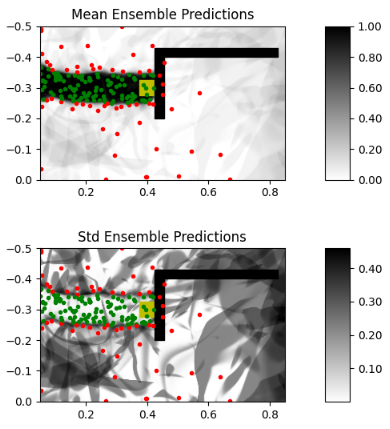
(c) Blue Block $\hat{f}_{pick}$



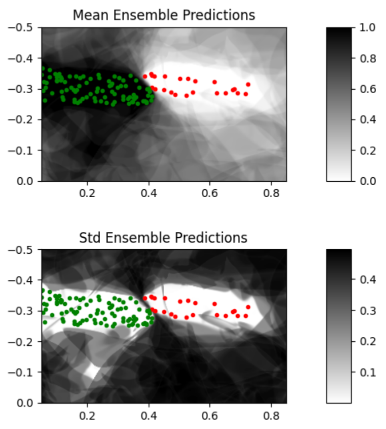
(d) Blue Block $\hat{f}_{holding}$



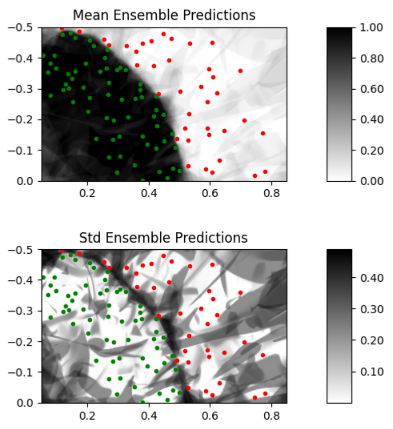
(e) Yellow Block $\hat{f}_{contact}$



(f) Yellow Block $\hat{f}_{contact}$



(g) Yellow Block $\hat{f}_{pick}$



(h) Yellow Block $\hat{f}_{holding}$

Figure 7-12: **Learned action feasibility model predictions for Sequential Goals.** These plots show the ensemble predictions of the AFMs learned from a single run of the Sequential Goals strategy. The grayscale color indicates the model's prediction for the given (x, y) input (see Section 7.3.2 for the model inputs). In the top mean prediction plots, a value of 1.0 indicates feasible action parameters, and 0.0 indicates infeasible action parameters. In the bottom standard deviation plots the value indicates the standard deviation of the ensemble's prediction for a given AFM input. The plotted points are the action parameters which were executed by the learning strategy and were found to be either feasible (green) or infeasible (red). For the $\hat{f}_{contact}$ AFMs we also visualize the contact configuration between the tool, shown in black, and the block, shown in its color in its initial position. From this contact configuration the robot attempts to move it to the different (x, y) positions in the plot. For the blue block $\hat{f}_{contact}$ AFM shown in (a), the robot is able to explore feasible pushes which move the block out of the tunnel (past $x = 0.4$). Now, when it attempts picking the blue block, it is occasionally successful. These pick attempts are shown in (c). The $\hat{f}_{contact}$ AFM shown in (b) is never feasible due to the tool colliding with the tunnel before it is ever able to reach the desired contact configuration. Once the robot has successfully picked up the blue block, all attempted `MoveHolding` actions are feasible, as shown in (d). For the yellow block the robot is able to see some positive labels for both $\hat{f}_{contact}$ AFMs shown in (a) and (b). Finally, both $\hat{f}_{pick}$ and $\hat{f}_{holding}$ learn how the weight of the yellow block impacts the success of the `Pick` and `MoveHolding` actions.

7.4.3 Plan Success

During the evaluation-phase, the robot finds a plan which achieves a given goal state, $s_{goal} \in \mathcal{S}_{goal}$, according to the following objective

$$a_{1:N} = \underset{a_{1:N} \in P_{goal}(s_{goal})}{\text{argmax}} \text{PF}(s_0, a_{1:N}) \mathbb{1}_{\{s=s_{goal}\}}(s_N),$$

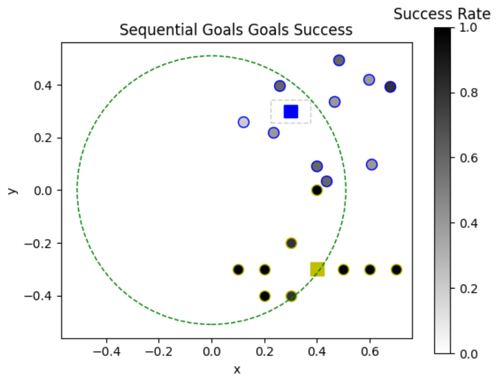
where $P_{goal}(s_{goal})$ consists of plans which achieve the goal state using the different plan skeletons given in Section 7.3.1. To evaluate the usefulness of our learning AFMs after executing and learning from 2500 actions, each AFM is given a set of 19 feasible goals from $\mathcal{S}_{goal}$. The success rate of planning to achieve these goals is shown in Table 7.1 for each method. While Sequential Goals does not perform perfectly, it is still markedly better than the baselines. Many of the learned models for Random Goals, Sequential Actions, and Random Actions were unable to find the small feasible subspace for pushing the blue block. This region opens up the robot to being able to reach a wide range of goal poses for the blue block. This accounts for most of the difference in performance between Sequential Goals and the baselines. Figure 7-13 shows the 19 goal states, and the success rate for each method. You can see that only the Sequential Goals method is able to achieve the goals for the blue block.

Method	**Plan Success**
Sequential Goals	0.73 ±0.44
Random Goals	0.48 ±0.50
Sequential Actions	0.48 ±0.50
Random Actions	0.20 ±0.40

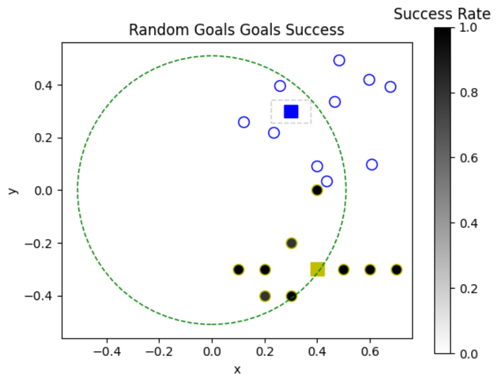
Table 7.1: **Planning performance of Sequential Goals and all baselines.**

7.5 Related Work

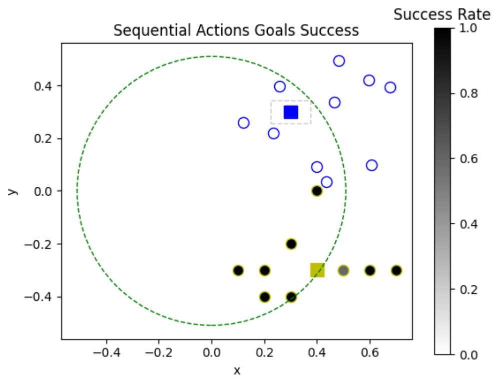
Tool use and pushing tasks are frequently used in robotic manipulation research due to the modeling challenges that they pose. These tasks often require fine-grained manipulation, complex objects and contact configurations, and dynamic interactions.



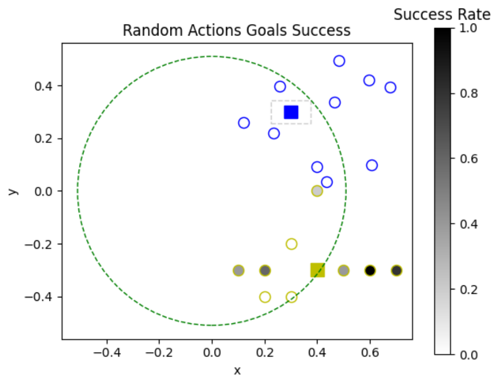
(a) Sequential Goals



(b) Random Goals



(c) Sequential Actions



(d) Random Actions

Figure 7-13: **Goals success rate for all methods.** These plots visualize how successful each method was at planning to achieve the evaluation-phase goals. The area which the robot is unable to carry the yellow block is shown in green and the tunnel is shown in gray. The initial block positions are given by the yellow and blue squares. Each evaluation-phase goal position and block is represented by a circle where the edge color indicates which block the goal is for, and the interior color of the circle is the success rate of the robot attempting to achieve the given goal. For each method we performed 5 runs, and use the resulting learned AFMs to plan to achieve the given goals. The Sequential Goals method is the only one which is able to achieve any goals for the blue block. This is due to its ability to learn the feasible subspace in the optimistic pushing model for the blue block. Once it learns this feasible subspace, it can find plans which first push the blue block out of the tunnel, then pick it up and move it elsewhere. The other methods are able to learn the yellow block AFMs with varying success.

Methods for learning dynamics models for pushing, poking, ball bouncing [11, 3, 2], and tool use [108, 44, 100] have been explored. Holladay et al. [44] give a method for selecting the highly constrained variables needed for accurately using tools such as screwdrivers, hammers, wrenches, and knives. Toussaint et al. [100] gives a formulation of a TAMP system which is able to solve for dynamic interactions by modeling the domain in a fully differentiable manner.

7.6 Conclusion

In this section we gave the Sequential Goals strategy which built upon the Sequential Actions strategies explored in Chapter 6. Sequential Goals provides a way for a robot to leverage a given space of goal states to plan towards while generating candidate plans for active learning. By using goal directed plans, as opposed to random actions, the robot is able to generate a rich search space for learning action feasibility models efficiently. We demonstrated the Sequential Goals strategy in a tool use domain where the size of the feasibility subspace within the optimistic action space is very small and difficult to model accurately.

Chapter 8

Conclusion

We conclude by discussing problems unaddressed by this thesis in Section 8.1, considering future work in Section 8.2, and finally summarizing the work presented in this thesis in Section 8.3.

8.1 Unaddressed Problems

We make various assumptions in our methods for learning task and action models. Primarily that,

- **As humans, we can hand-design optimistic models useful for learning.** The criteria for an optimistic model is that, for a single action step, the true underlying resulting states (feasible subspace) are a subset of the states the optimistic model believes the system can reach (optimistic space). This is a very loose constraint, and the size of the feasible subspace withing the optimistic space has an impact on the success of the methods outlined in this work. As action models become more optimistic, the size of the feasible subspace within the optimistic space grows proportionally smaller, making it more challenging to randomly sample.

- **Learning a classifier on top of optimistic models is sufficient for achieving evaluation-phase tasks.** When the outcome of an optimistically modelled action is different from the predicted outcome, our method provides no way of capturing this outcome to then use later in a planner. For example, if a robot is attempting a `MoveHolding` action, but the object falls out of the robot's grasp on the way, then our method would simply learn that those specific action parameters do result in successful `MoveHolding` actions. This could prevent the robot from ever learning alternative outcomes which might be useful later.

- **Exploring either random optimistic plans, or goal-directed plans for goals randomly sampled from $\mathcal{S}_{goal}$, is sufficient for discovering feasible actions.** We assume that these strategies will successfully lead the robot to attempt feasible actions. However, in some cases it may be nearly impossibly to find feasible action parameters, particularly for longer plans or very optimistic models.

- **It is safe to explore the space of an optimistically defined action model.** Exploring the space of an optimistic model is potentially dangerous. In our optimistically defined tool use task, the robot collides with the unmodelled tunnel when attempting to directly pick up the blue block inside of the tunnel. While this is a safe plan in simulation, in the real world it could potentially damage the robot. Optimistic exploration in the real world could also put people in harm's way.

- **As humans, we can hand-design a useful space of goals, $\mathcal{S}_{goal}$, to guide exploration.** To increase the ability of our method to generalize to various goals, which is the ultimate objective of robotic manipulation, $\mathcal{S}_{goal}$ might be difficult to specify by-hand in a way that allows the robot to later accomplish various tasks.

8.2 Future Work

Here we propose future work and potential solutions to the unaddressed problems previously stated.

- **As humans, we can hand-design optimistic models useful for learning.** We could explore various strategies for discovering both the best set of optimistic actions, and the best level of optimism for those actions. The effectiveness of a chosen optimistic model could be evaluated based on how effective it is at enabling the robot to achieve evaluation-phase tasks. When optimistic models are defined using PDDL, preconditions (or features), such as requiring objects to be in contact, are used to constrain the optimistic actions and ensure that the robot is able to quickly find feasible actions. However, there are many other types of actions and features we could bake into our optimistic models. For example, designing an optimistic action for using a door handle could be done in many different ways. A robot could explore at the level of small movements and learn to chain them together for successful door-twisting interactions, or it could explore the space of longer twisting actions designed by an engineer. All of these choices greatly impact the success of these learning methods for robotic manipulation. Automating this process could be an interesting future direction.

- **Learning a classifier on top of optimistic models is sufficient for achieving evaluation-phase tasks.** It could be useful to capture the alternative outcomes of optimistic actions; or actions which do not fall under the capabilities of the optimistic model. For example, we could cluster the alternative outcomes to learn additional models which might be useful during the evaluation-phase.

- **Exploring either random optimistic plans, or goal-directed plans for goals randomly sampled from $\mathcal{S}_{goal}$, is sufficient for discovering feasible actions.** To ensure that the robot is able to explore feasible actions in scenarios where that is increasingly difficult, it might be useful to leverage expert demonstrations [4]. These demonstrations would guide the robot to the feasible subspace and improve learning efficiency, as the robot would not have to waste time searching for the feasible subspace. However, expert demonstrations also impart bias on the learned models, so ways of balancing random exploration and expert demonstrations might be necessary.

- **It is safe to explore the space of an optimistically defined action model.** Expert demonstrations [4] could also be used to constrain exploration, and prevent the robot from attempting potentially dangerous actions. Safety could also be addressed through the use of safer control methods, such as impedance control [66], which allows a robot to quickly react to contact forces, preventing harm to people or itself. Soft robotics [71] has also been an increasingly active area of research, and addresses safety concerns through the use of compliant end effectors.

- **As humans, we can hand-design a useful space of goals, $\mathcal{S}_{goal}$, to guide exploration.** To alleviate the human bias of goal specification, we could allow the robot to generate its own useful space of goals from within the state space, to discover which learning-phase goals enable the best task completion during the evaluation-phase. It could also determine goal utility based on the information gained from executing various goal-directed plans.

8.3 Summary

In this thesis we have demonstrated several active learning strategies for learning both task feasibility models, $\hat{R}_{task}$, and action feasibility models, f_A. We have explored how visual state representations can be useful for generalizing to novel states via a learned GP prior, enabling quick task reward maximization in a constrained mechanism domain. We investigated active learning in sequential domains by extending the Bayesian Active Learning by Disagreement [45] strategy. In a block stacking domain this was performed with the help of a useful action abstraction and the Sequential Actions method. Finally, we improved upon Sequential Actions with the Sequential Goals method which enables a robot to generate a richer optimization search space consisting of goal-directed plans.

The key takeaways from the work in this thesis are that:

- Optimistic models are useful for constraining the exploration space of a robot while not eliminating potentially useful actions;

- Action feasibility can be integrated into an active learning objective in order to ensure that we consider both information gain and a robot's ability to execute actions;
- Generating an optimization search space for sequential manipulation problems is improved by using goal-directed plans.

Bibliography

[1] Ben Abbatematteo, Stefanie Tellex, and George Konidaris. Learning to generalize kinematic models to novel objects. In *PMLR Conference on Robot Learning (CoRL)*, 2019.

[2] Pulkit Agrawal, Ashvin V. Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[3] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie Pack Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[4] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. In *Robotics and Autonomous Systems*, volume 57, pages 469–483, 2009.

[5] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. In *Journal of Machine Learning Research*, volume 3, pages 397–422, 2002.

[6] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. In *Machine learning*, volume 47, pages 235–256, 2002.

[7] Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. In *Robotics and Autonomous Systems*, volume 61, pages 49–73, 2013.

[8] Patrick R. Barragán, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Interactive Bayesian identification of kinematic mechanisms. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[9] Peter W. Battaglia, Jessica B. Hamrick, and Joshua B. Tenenbaum. Simulation as an engine of physical scene understanding. In *Proceedings of the National Academy of Sciences (PNAS)*, 2013.

[10] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[11] Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[12] William H Beluch, Tim Genewein, Andreas Nurnberger, and Jan M Köhler. The Power of Ensembles for Active Learning in Image Classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] Niklas Bergström, Carl Henrik Ek, Mårten Björkman, and Danica Kragic. Scene understanding through autonomous interactive perception. In *International Conference on Computer Vision Systems (ICVS)*, 2011.

[14] Manuel Blum, Arnold Griffith, and Bernard Neumann. A stability test for configurations of blocks. In *Massachusetts Institute of Technology, Artificial Intelligence, Project MAC, Memo No. 188*, 1970.

[15] Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. In *The International Journal of Robotics Research*, volume 28, pages 104–126, 2009.

[16] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. In *International Conference on Learning Representations (ICLR)*, 2017.

[17] Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[18] Erwin Coumans and Yunfei Bai. Pybullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[19] Aidan Curtis, Minjian Xin, Dilip Arumugam, Kevin Feigelis, and Daniel Yamins. Flexible and efficient long-range planning through curious exploration. In *PMLR International Conference on Machine Learning (ICML)*, 2020.

[20] Varsha Dani, Thomas P. Hayes, and Sham M. Kakade. Stochastic linear optimization under bandit feedback. In *Conference on Learning Theory (COLT)*, 2008.

[21] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *Workshop on Safety, Security & Rescue Robotics (SSRR)*, 2009.

[22] Danny Driess, Ozgur Oguz, Jung-Su Ha, and Marc Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[23] Clemens Eppner, Roberto Martín-Martín, and Oliver Brock. Physics-based selection of actions that maximize motion for interactive perception. In *Worksop on Revisiting Contact - Turning a Problem into a Solution (RSS)*, 2017.

[24] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[25] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[26] Marshall L Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral combinatorics*, pages 73–87. Springer, 1978.

[27] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *International Conference on Learning Representations (ICLR)*, 2016.

[28] Fadri Furrer, Martin Wermelinger, Hironori Yoshida, Fabio Gramazio, Matthias Kohler, Roland Siegwart, and Marco Hutter. Autonomous robotic stone stacking with online next best object target pose planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[29] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *PMLR International Conference on Machine Learning (ICML)*, 2016.

[30] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian active learning with image data. In *PMLR International Conference of Machine Learning (ICML)*, 2017.

[31] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. In *The International Journal of Robotics Research*, volume 37, pages 104–136, 2018.

[32] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sampling-based methods for factored task and motion planning. In *The International Journal of Robotics Research*, volume 37, pages 1796–1825, 2018.

[33] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[34] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. In *Pattern Recognition*, volume 47, pages 2280–2292, 2014.

[35] Oliver Groth, Fabian B. Fuchs, Ingmar Posner, and Andrea Vedaldi. Shapestacks: Learning vision-based physical intuition for generalised object stacking. In *European Conference on Computer Vision (ECCV)*, 2018.

[36] Vijaykumar Gullapalli, Judy A. Franklin, and Hamid Benbrahim. Acquiring robot skills via reinforcement learning. In *IEEE Control Systems Magazine*, volume 14, pages 13–24, 1994.

[37] Abhinav Gupta, Alexi A. Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision (ECCV)*, 2010.

[38] Jung-Su Ha, Danny Driess, and Marc Toussaint. A probabilistic framework for constrained manipulations and task and motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[39] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The provable virtue of laziness in motion planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[40] Jessica B. Hamrick, Kelsey Rebecca Allen, Victor Bapst, Tina Zhu, Kevin R. McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. In *the Annual Meeting of the Cognitive Science Society (CogSci)*, 2018.

[41] Karol Hausman, Scott Niekum, Sarah Osentoski, and Gaurav S. Sukhatme. Active articulation model estimation through interactive perception. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[42] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations (ICLR)*, 2018.

[43] David Heckerman, John S. Breese, and Koos Rommelse. Troubleshooting under uncertainty. In *Microsoft Research Technical Report 94-07*, 1994.

[44] Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. Force-and-motion constrained planning for tool use. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[45] Niel Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. In *Workshop on Bayesian optimization, experimental design and bandits: Theory and applications (NeurIPS)*, 2011.

[46] Tadanobu Inoue, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana. Deep reinforcement learning for high precision assembly tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[47] Zhaoyin Jia, Andrew C. Gallagher, Ashutosh Saxena, and Tsuhan Chen. 3D Reasoning from Blocks to Stability. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2015.

[48] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Workshop on Bridging the gap between task and motion planning (AAAI)*, 2010.

[49] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. In *The International Journal of Robotics Research*, volume 32, pages 1194–1227, 2013.

[50] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Learning composable models of parameterized skills. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[51] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Gaussian processes for object categorization. In *International Journal of Computer Vision*, volume 88, pages 169–188, 2010.

[52] Dov Katz and Oliver Brock. Manipulating articulated objects with interactive perception. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[53] Beomjoon Kim, Leslie Kaelbling, and Tomás Lozano-Pérez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[54] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Adversarial actor-critic method for task and motion planning problems using planning experience. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[55] Beomjoon Kim and Luke Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. In *PMLR Conference on Robot Learning (CoRL)*, 2020.

[56] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation.

In *The International Journal of Robotics Research*, volume 38, pages 793–812, 2019.

[57] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F. Huber. A survey on learning-based robotic grasping. In *Current Robotics Reports*, 2020.

[58] James J. Kuffner and Steven M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[59] Alex Lagrassa, Steven Lee, and Oliver Kroemer. Learning skills to patch plans based on inaccurate models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[60] Fabien Lagriffoul and Benjamin Andres. Combining task and motion planning: A culprit detection problem. In *The International Journal of Robotics Research*, volume 35, pages 890–927, 2016.

[61] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. In *The International Journal of Robotics Research*, volume 33, pages 1726–1747, 2014.

[62] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. DeepMPC: Learning deep latent features for model predictive control. In *Robotics: Science and Systems (RSS)*, 2015.

[63] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. In *PMLR International Conference on Machine Learning (ICML)*, 2016.

[64] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. In *The Journal of Machine Learning Research*, volume 17, pages 1334–1373, 2016.

[65] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. In *The International Journal of Robotics Research*, volume 37, pages 421–436, 2018.

[66] Sheng-Yen Lo, Ching-An Cheng, and Han-Pang Huang. Virtual impedance control for safe human-robot interaction. In *Journal of Intelligent & Robotic Systems*, volume 82, pages 3–19, 2016.

[67] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[68] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[69] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *PMLR Conference on Robot Learning (CoRL)*, 2019.

[70] David J. C. MacKay. Information-based objective functions for active data selection. In *Neural Computation*, 1992.

[71] Mariangela Manti, Vito Cacucciolo, and Matteo Cianchetti. Stiffening in soft robotics: A review of the state of the art. In *IEEE Robotics & Automation Magazine*, volume 23, pages 93–106, 2016.

[72] Luis Montesano and Manuel Lopes. Learning grasping affordances from local visual descriptors. In *IEEE Conference on Development and Learning*, 2009.

[73] Caris Moses, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Learning to plan with optimistic action models. In *Workshop on Scaling Robot Learning (ICRA)*, 2022.

[74] Caris Moses, Michael Noseworthy, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Visual prediction of priors for articulated object interaction. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[75] Rémi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. In *Foundations and Trends in Machine Learning*, 2014.

[76] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[77] Anh Nguyen, Dimitrios Kanoulas, Darwin G. Caldwell, and Nikos G. Tsagarakis. Detecting object affordances with convolutional neural networks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[78] Michael Noseworthy, Caris Moses, Isaiah Brand, Sebastian Castro, Leslie Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Active learning of abstract plan feasibility. In *Robotics: Science and Systems (RSS)*, 2021.

[79] Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *PMLR International Conference on Machine Learning (ICML)*, 2017.

[80] Stefan Otte, Johannes Kulick, Marc Toussaint, and Oliver Brock. Entropy-based strategies for physical exploration of the environment's degrees of freedom. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[81] Pierre-Yves Oudeyer, Frédéric Kaplan, Verena V. Hafner, and Andrew Whyte. The playground experiment: Task-independent development of a curious robot. In *AAAI Spring Symposium on Developmental Robotics*, 2005.

[82] Matteo Papini, Alberto Maria Metelli, Lorenzo Lupo, and Marcello Restelli. Optimistic policy optimization via multiple importance sampling. In *PMLR International Conference on Machine Learning*, 2019.

[83] Hanna M. Pasula, Luke S. Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. In *Journal of Artificial Intelligence Research (JAIR)*, volume 29, pages 309–352, 2007.

[84] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[85] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *PMLR International Conference on Machine Learning (ICML)*, 2019.

[86] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[87] Lawrence G. Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[88] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter W Battaglia. Graph networks as learnable physics engines for inference and control. In *PMLR International Conference on Machine Learning (ICML)*, 2018.

[89] Yasser Shoukry, Pierluigi Nuzzo, Indranil Saha, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. Scalable lazy smt-based motion planning. In *IEEE Conference on Decision and Control (CDC)*, 2016.

[90] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *PMLR International Conference on Machine Learning (ICML)*, 2019.

[91] Tom Silver, Kelsey Rebecca Allen, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. Residual policy learning. In *arXiv preprint arXiv:1812.06298*, 2018.

[92] Tom Silver, Rohan Chitnis, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[93] Sanjiv Singh and Alonzo Kelly. Robot planning in the space of feasible actions: two examples. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[94] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *PMLR International Conference on Machine Learning (ICML)*, 2010.

[95] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[96] Siddharth Srivastava, Lorenzo Riano, Stuart Russell, and Pieter Abbeel. Using classical planners for tasks with continuous operators in robotics. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

[97] Jürgen Sturm, Cyrill Stachniss, and Wolfram Burgard. A probabilistic framework for learning kinematic models of articulated objects. In *Journal of Artificial Intelligence Research (JAIR)*, volume 41, pages 477–526, 2011.

[98] István Szita and András Lőrincz. The many faces of optimism: a unifying approach. In *PMLR International Conference on Machine Learning (ICML)*, 2008.

[99] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2015.

[100] Marc Toussaint, Kelsey Rebecca Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems (RSS)*, 2018.

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

[102] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[103] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Active model learning and diverse action sampling for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[104] Andrew M. Wells, Neil T. Dantam, Anshumali Shrivastava, and Lydia E. Kavraki. Learning feasibility for task and motion planning in tabletop environments. In *IEEE Robotics and Automation Letters (RA-L)*, volume 4, 2019.

[105] Patrick Winston. The MIT robot. In *Machine Intelligence*, volume 7, 1972.

[106] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.

[107] Victoria Xia, Zi Wang, Kelsey Rebecca Allen, Tom Silver, and Leslie Pack Kaelbling. Learning sparse relational transition models. In *International Conference on Learning Representations (ICLR)*, 2018.

[108] Danfei Xu, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[109] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. In *AI Open*, volume 1, pages 57–81, 2020.