Autonomously Learning World Model Representations For Efficient Robot Planning

by

Naman Shah

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2024 by the
Graduate Supervisory Committee:

Siddharth Srivastava, Chair
Subbarao Kambhampati
George Konidaris
Alberto Speranzon
Yu Zhang

ARIZONA STATE UNIVERSITY

May 2024

ABSTRACT

In today's world, robotic technology has become increasingly prevalent across various fields such as manufacturing, warehouses, delivery, and household applications. Planning is crucial for robots to solve various tasks in such difficult domains. However, most robots rely heavily on humans for world models that enable planning. Consequently, it is not only expensive to create such world models, as it requires human experts who understand the domain as well as robot limitations, these models may also be biased by human embodiment, which can be limiting for robots whose kinematics are not human-like.

This thesis answers the fundamental question: Can we learn such world models automatically? This research shows that we can learn complex world models directly from unannotated and unlabeled demonstrations containing only the configurations of the robot and the objects in the environment.

The core contributions of this thesis are the first known approaches for $i)$ task and motion planning that explicitly handle stochasticity, $ii)$ automatically inventing neuro-symbolic state and action abstractions for deterministic and stochastic motion planning, and $iii)$ automatically inventing relational and interpretable world models in the form of symbolic predicates and actions.

This thesis also presents a thorough and rigorous empirical experimentation. With experiments in both simulated and real-world settings, this thesis has demonstrated the efficacy and robustness of automatically learned world models in overcoming challenges, generalizing beyond situations encountered during training.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support and guidance of Prof. Siddharth Srivastava, my PhD supervisor. I am indebted to his unwavering support, invaluable guidance, and mentorship throughout this journey. Siddharth's diligence and focus on details has been a role model for me. His research methodologies have been pivotal in shaping values of mathematically formulating my ideas.

I extend my heartfelt appreciation to the members of my thesis committee, Prof. Subbarao Kambhampati, Prof. George Konidaris, Dr. Alberto Speranzon, and Prof. Yu Zhang for their insightful feedback, constructive criticism, and valuable suggestions that have significantly enhanced the quality of this thesis.

I have been fortunate to undertake three remarkable internships during my PhD journey. I extend my heartfelt gratitude to all my mentors during these internships: Prof. Georgios Fainekos, Dr. Bardh Hoxha, Dr. Andreas Kolling, Dr. Matthew Klenk, and Dr. Shiwali Mohan. Their guidance has been invaluable in providing me with industry experience crucial for my academic growth.

To my dear friends, I would not have been able to survive this journey without you all. Rashmeet, Pulkit, Kevin, Maitry, and Akku, I thank you all for having my back all these years and for being my chosen family. Our game nights on Fridays kept me going for all these years rejuvenating me for the research every week. I want to specially thank Rashmeet for her constant support and for being such a great friend. I would also like to thank my friends away from ASU – Harshil, Vrushabh, Kushal, Reeya, and Yash. Furthermore, I also want to thank friends that I made during my professional journey: Pablo, Akhil, Hardik, Ankit, David.

My heartfelt thanks go to all my colleagues at the AAIR Lab for fostering a conducive research environment and providing access to essential resources. I am particularly grateful to Nicholas Beck, Pamela Dunn, and Monica Dugan for their technical expertise and

administrative assistance.

I wish to acknowledge the National Science Foundation (NSF) for their financial support, without which this research project would not have been feasible.

Finally, I dedicate this thesis to my family for their boundless love, unwavering encouragement, and steadfast belief in my abilities. Their sacrifices and understanding have been a constant wellspring of motivation and strength throughout this journey.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER

1 INTRODUCTION 1

1.1 Approach and Thesis Scope 3

1.2 Contributions of Each Chapter 5

1.3 List of Discussed Papers 8

2 FORMAL FRAMEWORK 10

2.1 The Robot Planning Problem 10

2.2 The Motion Planning Problem 13

2.3 Symbolic Abstraction of a Continuous Robot Planning Problem 14

2.4 Conclusion 18

3 STOCHASTIC TASK AND MOTION PLANNING 19

3.1 Entity Abstraction 20

3.2 Computing Task and Motion Policies 21

3.2.1 HPlan for STAMP 24

3.3 Empirical Evaluation 30

3.3.1 Analysis of the results 34

3.4 Related Work 37

3.4.1 Stochastic Task Planning 37

3.4.2 Hierarchical Planning 38

3.4.3 Motion Planning 40

3.4.4 Integrated Task and Motion Planning 40

3.5 Conclusion 42

CHAPTER Page

4 AUTOMATICALLY LEARNING ZERO-SHOT ABSTRACTIONS FOR DETERMINISTIC MOTION PLANNING ... 43
4.1 Critical Regions ... 44
4.1.1 Learning to Predict Critical Regions ... 44
4.2 Inventing State and Action Abstractions ... 48
4.3 Hierarchical Abstraction-Guided Robot Planner (HARP) ... 51
4.4 Empirical Evaluation ... 56
4.4.1 Analysis of the Results ... 58
4.5 Related Work ... 64
4.6 Conclusion ... 67
5 ZERO-SHOT OPTION INVENTION FOR STOCHASTIC MOTION PLANNING PROBLEMS ... 69
5.1 Zero-Shot Option Inventors ... 70
5.1.1 Zero-Shot Option Endpoints ... 71
5.1.2 Zero-Shot Option Guides ... 73
5.2 Hierarchical Stochastic Motion Planning Using Zero-Shot Options ... 74
5.2.1 Theoretical Results ... 76
5.3 Empirical Results ... 81
5.3.1 Analysis of Results ... 84
5.4 Related Work ... 87
6 AUTOMATICALLY INVENTING RELATIONAL WORLD MODELS FOR ROBOT PLANNING ... 92
6.1 Relative Poses ... 93
6.2 Learning World Models ... 94

6.2.1 Inventing Predicates 95
6.2.2 Inventing Symbolic Actions 101
6.3 Planning with Learned Abstractions and Continual Learning 106
6.4 Empirical Evaluation 109
6.4.1 Analysis of Results 112
6.5 Related Work 116
7 OTHER APPLICATIONS 145
7.1 Motivation 145
7.2 Architecture 147
7.3 Conclusion 150
8 CONCLUSION AND FUTURE WORK 151
REFERENCES 154

LIST OF TABLES

Table Page

3.1 Critera defined by Lagriffoul *et al.* (2018) evaluated in each of the test domains. 32

3.2 Summary of times taken to solve the STAMP problems. Timeout: 4000 seconds. 35

5.1 Percentage of options that our approach reused from the task they were computed to every subsequent task they were needed across 5 test tasks in each environment. 87

6.1 Detailed statistics about the empirical evaluation and invented abstractions. The success rate is an average of 50 independent test tasks with 5 random seeds. 114

6.2 Ablation study of our approach with decreased training demonstrations and comparison with baseline approaches. The success rate for our approach and baselines averaged over 10 different unseen test tasks and 5 random executions. The percentages represent the fraction of training demonstrations used for learning the initial state and action abstractions. 115

LIST OF FIGURES

Figure Page

2.1 An example of a robot planning problem. The environment consists of an object o of type `can` and a gripper g of type `robot`. The figure on the left shows an initial state and the figure on the right shows a goal state. 12

2.2 Specification of an abstract action that picks up an object (obj_1) using a robot $(gripper_1)$. 17

3.1 Left: YuMi robot uses HPlan (Sec. 3.1) to build a 3π structure (2 π structures stacked on top of a π structure) with Keva planks despite uncertainty in their initial locations. Right: A stochastic variant of the cluttered table domain where the robot is instructed to pick up the black can, but pickups may fail and crush the cans requiring them to be disposed of. 19

3.2 Plan refinement graph (PRG) used to maintain separate abstract models. Each plan refinement node (PRN) contains an abstract model, partially refined policy, and current state of refinement. Each edge contains refinement for a partial policy (σ_{ij}) and a failure reason (p_k). 22

3.3 Left: Backtracking from node B invalidates the concretization of subtree rooted at A. Right: Replanning from node B 25

Figure Page

3.4 A working example for Alg. 1. (a) shows the initial environment configuration. The goal for the robot is to pick up the “Red” object which is surrounded by “Blue”, “Green”, “Orange”, and “Black” objects. G is the end-effector of a robot. (b) shows a high-level, abstract task specification of the “pick” action. (c) shows the policy refinement graph (PRG) which is generated incrementally by Alg. 1. Each green box represents a policy refinement node (PRN). The tree in each PRN represents a high-level policy. Each node in a high-level policy is a state-action pair. For brevity, we only show high-level action in the node. Trees with dotted lines are partial policies. The red number represents p/c ratio for each RTL path in a policy. 26

3.5 Top: Cluttered Table: The Fetch mobile manipulator uses a STAMP policy to pick up a target bottle while avoiding those likely to be crushed. It replaces a bottle that was not crushed (left), discards a bottle that was crushed (center) and picks up the target bottle (right). Bottom: Building Structures with Keva Planks: ABB YuMi builds Keva structures using a STAMP policy: 12-level tower (left), twisted 12-level tower (center), and 3-towers (right). 27

3.6 Top: Aircraft Inspection: UAV inspects faulty parts of an aircraft in an airplane hangar and alerts the human about the location of the fault. UAV’s movements and sensors are noisy, so it may drift from its location or fail to locate the fault. Bottom: Find the can: Fetch searches for a can in drawers. The can can be placed in one of the drawers stochastically. 28

Figure Page

3.7 Setting up a dining table: Fetch uses STAMP policy to set up a dining table. A tray is available to carry multiple items at a time but carrying more than two items on the tray may break the items. Left: The initial state. Right: The goal state. 31

3.8 Anytime performance of ATM-MDP, showing the time in seconds (x-axis) vs. probability mass refined (y-axis). 34

4.1 (a) An illustrative environment for a motion planning problem. The robot (R) is tasked to reach the kitchen (K). Red regions (b) show predicted critical regions in the environment. Lastly, (c) shows a projection of the computed state abstraction. Each colored cell represents an abstract state. Arrows show examples of abstract actions, defined as transitions between abstract states. 43

4.2 Our overall network architecture. Our approach uses the number of degrees of the robot to automatically generate a network architecture that can be used for learning to predict critical regions in unseen test environments. 45

4.3 Test environments for our approach. Dimensions of environments (A)-(D) are $\mathbf{5m} \times \mathbf{5m}$ and dimensions of environments (E)-(G) are $\mathbf{25m} \times \mathbf{25m}$. (A)-(G) are used for navigational problems while (H) and (I) are used for manipulation problems. 56

Figure Page

4.4 Predicted critical regions and computed state abstractions for 3-DOF rectangular (R) and Car robots. (a) Input to the environment. (b) Critical regions for the location of the robot's base link in the workspace. (c) Critical regions for the rotation of the robot's base link of the robot. Blue regions are locations where the network predicted the robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical. (d) 2D projections of state abstraction generated by our approach. State abstractions are strictly for visualization as our approach does not require generating them. 59

4.5 Predicted critical regions and generated state abstractions for a 4-DOF hinged robot. (a) Input to the environment. (b) Critical regions for the location of the robot's base link in the workspace. (c) Critical regions for the orientation of the robot's base link. Blue regions are locations where the network predicted the robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical. (d) Critical regions for the hinge joint. Blue regions show that the network predicted it to be closer to $180°$ and green regions show that the network predicted the hinge angle close to $90°$ or $270°$. (e) 2D projections of state abstraction generated by our approach. State abstractions are strictly for visualization as our approach does not require to generate this explicitly. 60

Figure Page

4.6 Predicted critical regions for an 8-DOF Fetch robot. The green region in (a) shows the goal location for the end effector. (b) shows the critical regions generated by the learned model. Although the network predicts critical regions for all the joints, only critical regions or end-effector's location in the workspace are shown. 61

4.7 Each plot shows the fraction of **100** independently generated motion planning tasks solved (y-axis) in the given time (x-axis) for all the test environments and robots. The title of each subplot represents the robot and the environment. E.g., "R - A" stands for rectangular robot in environment A (Fig. 4.3(A)) and "H - A" stands for hinged robot in environment A. 62

4.8 (a) Solving **20** randomly generated problems repeatedly 1**10** times. The x-axis shows the problem iteration and the y-axis shows the average time over **20** problem instances. (b) Time taken to solve **100** randomly generated problem instances. The X-axis shows the problem number and the y-axis shows the taken to solve each problem. 63

Figure Page

5.1 Our overall approach for automatically inventing high-level options. (a) shows the input to our system. (b) shows the zero-shot abstraction process along with the raster scan of the input environment (left), critical regions predicted by the learned network (center), and the zero-shot state abstraction (right). The top image in (c) shows a subset of automatically invented interface options and the bottom image shows a subset of automatically invented centroid options. Lastly, these learned options are used for hierarchical planning and learning. The red arrows in (d) show an example of a high-level plan over centroid options given an initial configuration (orange area) and a goal configuration (green area). Policies for these options are learned using deep reinforcement learning and the auto-generated dense pseudo-reward function. 69

5.2 Our test environments and robots. 80

5.3 (Higher values are better) Times taken (averaged over 5 trials) by our approach (SHARP) and baselines to compute solutions in the test environments. The X-axis shows the time and the y-axis shows the fraction of the problems solved in the given time. 81

5.4 (Smaller bars and darker circles are better) An average number of steps taken in **successful** executions of the learned policies and success rates for our approach and the baselines. The pie chart over each bar represents the success rate (shaded black area) while executing the learned policy. 83

Figure | Page

5.5 Test environments of the size $15m \times 15m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions used to synthesize centroid options. Bottom: The white circles represent the interface regions for each pair of abstract states used to synthesize interface options.ere 85

5.6 Test environments of the size $75m \times 75m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions used to synthesize centroid options. Bottom: The white circles represent the interface regions for each pair of abstract states used to synthesize interface options. 86

6.1 Our overall approach. We start with a set of demonstrations on relatively simple tasks using a simple robot and learn a symbolic model in the form of a set of predicates and high-level actions. This symbolic model can be used with any off-the-shelf planner for solving unseen complex long-horizon planning problems with other similar robots. 92

6.2 An illustration for computing relative poses 93

6.3 An example of relational critical regions. The gripper g is tasked to grasp the object o. Stages 1 and 3 show an initial configuration of the gripper g and stages 2 and 4 show the final configuration. The inset figures show the pose of the gripper for stages 2 and 4 in the relative reference frame of the object and the blue region shows the identified relative critical region. 96
6.4 Trajectory $T_1 = \langle S_1', S_2', S_3', S_4' \rangle$ corresponding to the process of picking up a yellow cup from the table. The state description below each image explains that image in English. These state descriptions are added here for ease of understanding only. 104
6.5 Trajectory $T_2 = \langle S_1', S_2', S_3', S_4' \rangle$ corresponding to the process of picking up a green cup from the table. The state description below each image explains that image in English. These state descriptions are added here for ease of understanding only. 104
6.6 Different relations invented by our approach and their corresponding critical regions. Each image shows one binary predicate and its semantic interpretation. The red dots show sampled possible poses for the object in the relational critical region. 110
6.7 An automatically invented high-level action by our approach. The top figure shows the states before and after executing the high-level action. The bottom part shows the automatically learned precondition and effect of the high-level action. 112
7.1 JEDAI system with a Blockly-based plan creator on the left and a simulator window on the right. 146

Figure Page

7.2 Architecture of JEDAI showing interaction between the four core components. 147

Chapter 1

INTRODUCTION

Recent years have seen a sharp increase in the use of robots in various areas such as manufacturing, household robots, warehouses, delivery, and many more. These complex tasks require robots to take different actions that change the state of the robot and other objects in the environment. Akin to humans, robots also have to use their actions in a sequence to achieve these tasks and require planning and reasoning over a long horizon to come up with such sequences of actions.

Robot actions usually consist of low-level continuous actions that control the joints of the robots. Typically, sampling-based motion planners such as RRT (Lavalle, 1998) or PRM (Kavraki *et al.*, 1996) are used with such actions for composing them and achieving different tasks. However, there are two major drawbacks to using them for solving complex robot planning problems: 1) Motion planners cannot directly handle changing configuration spaces which occur when robots interact with objects in the environment; 2) Motion planners are designed to address large search spaces with continuous low-level actions and continuous states by sampling. However, these approaches do not scale when the planning horizon increases. These drawbacks restrict the use of motion planners for complex tasks where robots need to interact with different objects in the environment.

Hierarchical planning systems (e.g., task and motion planning (Cambon *et al.*, 2009; Srivastava *et al.*, 2014)) have been prepared for solving such tasks. Advancements in symbolic planners such as Fast-Forward (FF) (Hoffmann, 2001) and Fast-Downward (FD) (Helmert, 2006) allow efficiently solving symbolic planning problems. Task and motion planners use these planners for computing high-level, symbolic solutions and refine them using sampling-based motion planners into a sequence of actions that robots can execute.

However, such hierarchical planners require *world models* to be able to compute solutions for complex robot planning problems. These world models include symbolic predicates that induce state abstractions and abstract actions. E.g., for a warehouse setting, symbolic predicates would include predicates such as `AtLoc` and `Holding` that define the location of the objects and what object a robot is holding, and symbolic actions would include actions such as `Pick` or `Move` that either moves the robot from one location to another location or allows a robot to pick items.

One of the major criticisms of hierarchical planning systems has been the following: How should a robot develop these *world models* for planning? Today, typically, human experts provide these abstractions. However, this process requires a domain expert who is also familiar with the robot's constraints. Moreover, these abstractions are heavily biased by human intuition, and therefore, they usually only involve properties such as `On(A, B)` and `InGripper(R, A)` and actions such as `Pick` and `Place`, and are only restricted to human-like robots.

This thesis answers the above question by developing approaches that automatically *invent* such world models. We develop approaches that use unannotated and unsegmented low-level trajectories for learning symbolic and interpretable world models. These world models are highly generalizable, i.e., they are zero-shot transferrable to new unseen environments and problems. Additionally, we also develop approaches that efficiently use these automatically invented world models for scalable and long-horizon robot planning. We show the effectiveness and robustness of the invented world models through our theoretical and empirical results using a wide range of robots and environments in both simulated and real-world settings.

Recently, there has been advances in learning-based approaches (Devin *et al.*, 2017; Pathak *et al.*, 2019; Martín-Martín *et al.*, 2019; Huang *et al.*, 2020b; Shah *et al.*, 2023; Vuong *et al.*, 2023; Hafner *et al.*, 2023) that use end-to-end deep neural networks. Some

researchers have argued that these approaches implicitly learn abstract world models. However, these implicit world models are not interpretable and do not provide any guarantees of correctness. Additionally, most of these approaches focus on significantly smaller horizon problems compared to the problems this thesis aims to solve and require a huge amount of training experience on the test tasks as well as dense and carefully crafted reward structures, or expert demonstrations of these tasks. Conversely, such approaches can be as complimentary to the approaches developed as part of this thesis and can be used for learning low-level primitive controllers for learning low-level controllers for automatically invented world models.

Now, we discuss the scope of this thesis and the research directions advanced by it.

1.1 Approach and Thesis Scope

This thesis brings together various distant research areas of AI and robotics. In this section, we introduce each of these major research areas and highlight the contribution of this thesis in each of these areas. We also briefly discuss some state-of-the-art approaches in these research directions. However, we discuss most of the related research in the most relevant chapter.

Motion planning A motion planner computes a path that a robot can follow to reach a final *configuration* from an initial *configuration*. Due to the large state space, sampling-based motion planners are generally used to compute such motion plans. However, state-of-the-art sampling-based motion planners rely on unguided uniform sampling. This thesis introduces hierarchical motion planners that combine high-level symbolic planning with sampling-based motion planning or deep reinforcement learning for guiding the motion planning and efficiently computing motion plans while automatically inventing the hierarchies and reward functions. Through theoretical and empirical results, we show that the

novel hierarchical motion planners proposed in this thesis significantly outperform existing state-of-the-art sampling-based motion planners.

Abstractions Various abstraction methods have been long-used in automated planning. Earlier symbolic planning approaches such as Sacerdoti (1974) and Knoblock (1990) use symbolic abstractions for automatically learning heuristics. State-of-the-art symbolic planners such as FF (Hoffmann, 2001) and FD Helmert (2006) use predicate abstraction to relax the planning problem. Several approaches have used abstractions for efficient planning. Srivastava *et al.* (2001) use abstractions for efficiently solving a planning and scheduling problem. Abstractions have also been used with robot planning problems to convert them to symbolic planning problems. Marthi *et al.* (2007a) and Srivastava *et al.* (2016a) present foundational concepts for defining and using abstractions for robot planning problems. The central idea of this thesis is to use and automatically learn such abstractions for efficient robot planning.

Task and motion planning Abstractions allow robot planning problems to be solved using symbolic planners. However, robots cannot use these symbolic plans directly as they can only execute motion plans. Each symbolic action must be converted into an executable motion plan. A naíve approach would be to first compute a symbolic plan and then compute motion plans for actions in the plan. However, due to the lossy nature of the abstractions, not every high-level action can be refined into a motion plan. Therefore, task and motion planning approaches (Cambon *et al.*, 2009; Kaelbling and Lozano-Pérez, 2011; Srivastava *et al.*, 2014) search for a high-level plan that has motion planning refinements for every high-level action in the plan. Most of the recent task and motion planning approaches focus on deterministic motion planning problems. This thesis presents the first known anytime task and motion planning approach for stochastic robot planning problems, where a

robot's actions have multiple possible outcomes. Through our empirical evaluation, we show that the anytime property of the approach allows the robot to start executing the solution significantly faster without computing the full solution while avoiding the deadends that may arise from the determinization of the problem.

Learning world models Learning symbolic world models (or abstractions) for robot planning has been at the center of attention recently. This is also the main focus of this thesis. A task and motion planner requires abstractions in mainly three forms: $i)$ state abstractions in the form of symbols or predicates that represent relations between different objects in the environment and provide a finite state space, $ii)$ actions that the robot can perform in the form of symbolic high-level actions that provide a bounded branching factor for symbolic planning, and $iii)$ a set of *pose generators* that can be used to refine high-level actions into motion plans for the robot to execute. Konidaris *et al.* (2018) focus on inventing symbols for robot planning and learning symbolic representations for high-level robot skills (actions). Silver *et al.* (2022) present a complementary approach for learning high-level actions that the robot can execute. However, existing approaches require partial abstractions to be provided as input and do not learn all three components simultaneously. In this thesis, we present the first known approach for automatically inventing symbolic predicates and high-level actions simultaneously – from unannotated and unsegmented robot trajectories. We show that these learned abstractions are not only effective for hierarchical robot planning, but they are also robust and interpretable.

Now, we discuss the contribution of each chapter of the thesis.

1.2 Contributions of Each Chapter

Technical contributions made by this thesis are mainly divided into three blocks: (Ch. 3) We present the first known approach for solving stochastic task and motion planning prob-

lems, (Ch. 4 and Ch. 5) a novel approach for learning state and actions abstractions for deterministic and stochastic motion planning problems, and (Ch. 6) a novel approach for inventing relational world models that enable hierarchical robot planning. More details about related research are reported in the most relevant chapters.

Chapter 2: "Formal Framework" This chapter revisits some key concepts and provides examples of motion planning, robot planning, symbolic abstractions, and symbolic robot planning problems. It defines the core robot planning problem that various approaches developed as part of this dissertation solve. The material in this chapter draws upon joint work with Deepak Kala Vasudevan, Kislay Kumar, Pranav Khamojhalla, Abhyudaya Srinet, Jayesh Nagpal, Pulkti Verma, and Siddharth Srivastava (Shah *et al.*, 2020; Shah and Srivastava, 2021, 2022b, 2024; Shah *et al.*, 2024).

Chapter 3: "Stochastic Task and Motion Planning" This chapter tackles a relatively simpler problem of using hand-coded abstractions for robot planning and presents our novel anytime approach for solving stochastic task and motion planning problems. This is the first known approach that explicitly handles noise in the robot's actions while computing task and motion policies leading to policies that actively avoid deadends that may arise due to the stochastic nature of the system. At the same time, the anytime algorithm allows the robot to start executing the policy without having to wait for a complete solution. The material in this chapter is based on joint work with Deepak Kala Vasudevan, Kislay Kumar, Pranav Khamojhalla, and Siddharth Srivastava (Shah *et al.*, 2020; Shah and Srivastava, 2021).

Chapter 4: "Automatically Learning Zero-Shot Abstractions For Deterministic Motion Planning" As highlighted in Sec. 1.2, robot planning approaches (such as the one presented in Ch. 3) require humans to provide world models. However, these require human

experts. This chapter presents the first known approach for automatically inventing world models for motion planning problems in the form of neuro-symbolic state and action abstractions. We present an approach for automatically generating a robot-specific neural network architecture that predicts *critical regions* in unseen environments. Our approach uses these critical regions for automatically inventing state and action abstractions. Lastly, we present a novel hierarchical multi-source multi-directional planner that efficiently uses invented abstractions for sampling-based motion planning for holonomic and non-holonomic robots. We present strong theoretical results for holonomic robots and a strong empirical evaluation that shows improved performance by order of magnitude over state-of-the-art sampling-based motion planners. This work is based on joint work with Abhyudaya Srinet and Siddharth Srivastava (Shah *et al.*, 2021; Shah and Srivastava, 2022b).

Chapter 5: "Zero-Shot Option Invention for Stochastic Motion Planning Problems" Motion planning typically assumes perfect actuation for robots. However, in most real-world scenarios, robots do not possess perfect actuation. In such settings, robots require motion policies instead of motion plans. This chapter presents the first known approach for automatically inventing neuro-symbolic world models for stochastic motion planning problems. We present an approach that invents options in a zero-shot fashion, i.e., inventing options for unseen environments and problems while not requiring additional training. Our approach combines two very distinct paradigms of AI: symbolic planning and deep reinforcement learning. It provides strong theoretical guarantees for holonomic robots and significantly improved empirical performance over existing sampling-based motion planners and deep reinforcement learning approaches. The content in this chapter is based on joint work with Siddharth Srivastava (Shah and Srivastava, 2022a, 2024).

Chapter 6: "Automatically Inventing Relational World Models For Robot Planning" Real-world applications require robots to interact with different objects in the environment. This chapter presents a novel approach for automatically inventing relational abstractions in the form of symbolic predicates and high-level actions defined using invented predicates. This is the first known approach that only uses a small set of unannotated and unsegmented low-level robot trajectories for automatically and simultaneously inventing interpretable state and action abstractions that support hierarchical robot planning. Through empirical evaluation, we show that the automatically invented predicates and actions are interpretable, and enable efficient hierarchical robot planning. This chapter is based on joint work with Jayesh Nagpal, Pulkit Verma, and Siddharth Srivastava (Shah *et al.*, 2024).

Chapter 7: "Other Applications" This chapter presents JEDAI – an application of the approaches developed as part of this thesis designed to educate students and non-experts with concepts of AI planning and robotics. This material is based on joint work with Trevor Angle, Pulkit Verma, and Siddharth Srivastava (Shah *et al.*, 2022).

1.3 List of Discussed Papers

We now present the list of works in reverse chronological order discussed in this thesis.

1. **Shah, N.**, Nagpla, J., Verma, P., and Srivastava, S. 2024. From Reals to Logic and Back: Learning Symbols and Generalizable Representations for Long-Horizon Planning. (In submission)

2. **Shah, N.** and Srivastava, S., 2024. Hierarchical Planning and Learning for Robots in Stochastic Settings Using Zero-Shot Option Invention. In Proceedings of Association for the Advancement of Artificial Intelligence (AAAI), 2024.

3. **Shah, N.** and Srivastava, S., 2022. An Anytime Hierarchical Approach for Stochastic Task and Motion Planning (In submission)

4. **Shah, N.** and Srivastava, S., 2023. Learning to Create Abstraction Hierarchies for Motion Planning Under Uncertainty. In IJCAI 2023 Workshop on Bridging the Gap: Planning and Reinforcement Learning (PRL).

5. **Shah, N.** and Srivastava S., 2023. Learning Neuro-Symbolic Abstractions for Motion Planning Under Uncertainty. In IJCAI 2023 Workshop on Neuro-symbolic Agents (NSA).

6. **Shah, N.***, Verma, P.*, Angle, T., and Srivastava S., 2022. JEDAI: A System for Skill-Aligned Explainable Robot Planning. In Proceedings of International Foundation for Autonomous Agents and Multi-Agent Systems (AAMAS), 2022. (Demonstration track, **Best Demo Award**)

7. **Shah, N.** and Srivastava, S., 2022. Using Deep Learning to Bootstrap Abstractions for Robot Planning. In Proceedings of International Foundation for Autonomous Agents and Multi-Agent Systems (AAMAS), 2022.

8. **Shah, N.**, Srinet, A. and Srivastava, S., 2020. Learning and Using Abstractions for Robot Planning. In 2021 ICAPS Workshop on Planning and Robotics (PlanRob)

9. **Shah, N.**, Vasudevan, D.K., Kumar, K., Kamojjhala, P. and Srivastava, S., 2020. Anytime integrated task and motion policies for stochastic environments. In Proceedings of International Conference Robotics and Automation (ICRA), 2020. (Also appeared in ICAPS 2019 workshop on Planning and Robotics (PlanRob))

Chapter 2

FORMAL FRAMEWORK

This chapter defines the fundamentals to a robot planning problem. We first explain the notion of a robot and the configuration space and then define a robot planning problem in the continuous configuration space. We further relate a robot planning problem to the classical notion of a motion planning problem. Lastly, we describe an alternate representation of the continuous robot planning problem and the notion of symbolic abstractions.

2.1 The Robot Planning Problem

A robot is a kinematic chain of rigid-body links and joints (LaValle, 2006). Each pair of links is connected with a joint. Typically, the state of a robot is represented using a set of joint values for each joint of the robot. A forward kinematic function can be used to compute the poses of each point on the links given the geometry of the links and the joint values. Conversely, an inverse kinematic function can be used to compute the set of joint values for each joint given the target poses of the links.

We define the robot state-space $\mathcal{X}$ as a set of all configurations of the robot. Given a collision function c, this state-space can be partitioned such that $\mathcal{X} = \mathcal{X}_{\text{obs}} \cup \mathcal{X}_{\text{free}}$ where $\mathcal{X}_{\text{obs}}$ refers to the states where the robot is in a collision and $\mathcal{X}_{\text{free}}$ refers to the set of states where the robot is free of collisions.

An object o in a 3D environment is defined as a rigid body. The state of an object is defined as a 6D pose of an object. A 6D pose represents the translation and the rotation of an object with respect to a fixed frame of reference.

We now define a 3D environment for a robot planning problem as a first-order logic model. A first-order logic model consists of a universe and a vocabulary (a set of relations

or functions). The universe $\mathcal{O} = \{o_1, \ldots o_n, r_1, \ldots r_k\}$ is a set of typed objects and robots in the environment. Let $\mathcal{T}$ define the set of all object types.

We define the robot planning vocabulary $\mathcal{V} = \{P^{\mathcal{W}}, T\}$ as a set of two functions. $T : \mathcal{O} \to \mathcal{T} : o \mapsto t$ maps each object $o \in \mathcal{O}$ to an object type $t \in \mathcal{T}$. $P^{\mathcal{W}} : \mathcal{O} \to \mathbb{R}^6$ defines a 6D pose for each object $o \in \mathcal{O}$ in the world reference frame $\mathcal{W}$. Without the loss of generality, we overload $P^{\mathcal{W}}$ to define the configuration of the robot. Given an absolute or world reference frame $\mathcal{W}$, we refer to pose of an object o, ($P^{\mathcal{W}}(o)$), as $P_o^{\mathcal{W}}$. We omit the superscript $\mathcal{W}$ for brevity.

We assume a fully observable setting, i.e., states of all the objects (pose) and robots (configurations) are known. The state-space of a robot planning problem can be computed using the vocabulary $\mathcal{V}$. Without loss of generality, we refer to the set of all possible poses of the object o_i as $\mathcal{X}_{o_i}$ and the set of all possible configurations of the robot r_j as $\mathcal{X}_{r_j}$. The state-space for the robot planning problem is defined as a joint state-space and is represented as $\mathcal{X} = \bigoplus_{o_i} \mathcal{X}_{o_i} \bigoplus_{r_j} \mathcal{X}_{r_j}$ for every object $o_i \in \mathcal{O}$ and every robot $r_j \in \mathcal{O}$. Akin to the configuration space of the robot, the robot planning state space can also be partitioned into the set of collision-free states $\mathcal{X}_{\text{free}}$ and in-collision states $\mathcal{X}_{\text{obs}}$ such that $\mathcal{X} = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ given the collision function c. In the future, we will use $\mathcal{X}$ to refer to a robot planning state space (the joint state-space for robots and objects) when referring to a robot planning problem or to a configuration space of the robot when referring to a motion planning problem (defined is Sec. 2.2).

Primitive actions (low-level actions) are unit-cost and bounded actions that enable a robot to change its state, i.e., the configuration of the robot. It also changes the pose of any object that is grasped by the robot. This allows the robot to move around in the environment and manipulate different objects in the environment. Formally, let u_a define an intended control signal associated with a primitive action a. Each primitive action a defines a function $a : \mathcal{X} \to \mathcal{X} : x \mapsto \mu(x + u_a)$ where $\mu(x + u_a)$ is a probability measure at the intended

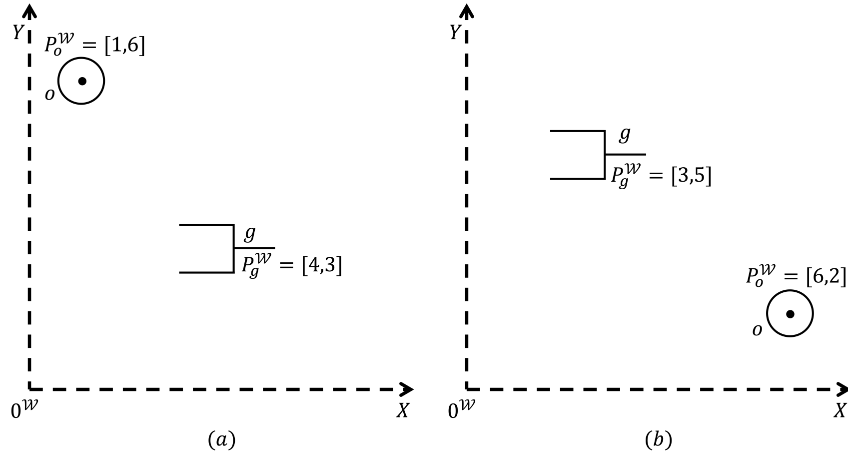


Figure 2.1: An example of a robot planning problem. The environment consists of an object o of type `can` and a gripper g of type `robot`. The figure on the left shows an initial state and the figure on the right shows a goal state.

target $x + u_a$ of the control action. Let $\mathcal{A}$ be the uncountably infinite set of primitive unit cost deterministic actions.

Now, we define a robot planning problem as a stochastic shortest path problem and use the set of actions as a transition function.

Definition 1 *A* ***robot planning problem*** *is defined as a tuple* $\mathcal{M} = \langle \mathcal{O}, \mathcal{T}, \mathcal{V}, \mathcal{A}, \mathcal{X}, x_i, \mathcal{X}_g \rangle$ *where,*

- $\mathcal{O}$ *is a set of objects and robots.*
- $\mathcal{T}$ *is a set of object types.*
- $\mathcal{V} = \{P, T\}$ *is a robot planning vocabulary.*
- $\mathcal{A}$ *is an uncountably infinite set of primitive robot actions.*
- $\mathcal{X}$ *is an environment state space.*
- $x_i \in \mathcal{X}$ *is an initial state.*
- $\mathcal{X}_g \subset \mathcal{X}$ *is a set of goal states.*

Example 1 *Consider a 2D environment shown in the Fig. 2.1. Fig 2.1(a) and (b) show initial and goal states for a robot planning problem. Here, the environment includes the object o_1 and a gripper g of object types* `can` *and* `robot` *respectively. Therefore,* $T(o_1) =$ `can` *and* $T(g) =$ `robot`*. In the Fig. 2.1(a),* $P_g^{\mathcal{W}} = [4,3]$ *and* $P_o^{\mathcal{W}} = [1,6]$ *show the 2D poses of the gripper g and object o respectively.*

A solution to a robot planning problem is a partial policy $\pi : \mathcal{X} \rightarrow \mathcal{A}$ that maps each reachable state (when starting with x_0 and executing the policy π) to a primitive action from the set of actions. Typically, various sampling-based methods can be used to compute such solutions. However, continuous action spaces and the infinite branching factor make it infeasible for primitive actions to be used for long-horizon robot planning problems. Therefore, a complex robot planning problem is either divided into multiple motion planning problems (defined in Sec. 2.2) or converted to a task and motion planning problem using abstractions (explained in Sec. 2.3) to be efficiently solved.

2.2 The Motion Planning Problem

The traditional notion of a motion planning problem is defined as a special case of the robot planning problem. A robot planning problem is a motion planning problem when the start state $x_i \in \mathcal{X}_{\text{free}}$ and the goal state $x_g \in \mathcal{X}_{\text{free}}$ are connected by a continuous path within a given configuration space of the robot. Formally, a motion planning problem for a robot configuration space $\mathcal{X}_r$ is defined as follows.

Definition 2 *A **motion planning problem** is a tuple* $\langle \mathcal{X}_r, \mathcal{A}, x_i, x_g, c \rangle$ *where,*

- $\mathcal{X}_r = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$ *is the configuration space of the robot.*
- $\mathcal{A}$ *is an uncountably infinite set of primitive robot actions such that every action* $a \in \mathcal{A}$ *defines a function* $a : \mathcal{X} \rightarrow \mathcal{X} : x \mapsto \mu(x + u_a)$ *where x is a configuration of the robot and u_a is an intended control for the action.*

- $c : \mathcal{X} \to \{T, F\}$ *is a collision function.*

- $x_i \in \mathcal{X}_{free}$ *is a collision-free initial configuration.*

- $x_g \in \mathcal{X}_{free}$ *is a collision-free goal configuration.*

Example 2 *Consider the environment shown in Fig. 2.1. An example of a motion planning problem would be a robot planning problem such that the initial state is where* $P_g^{\mathcal{W}} = [4, 3]$ *and the goal state is where* $P_g^{\mathcal{W}} = [3, 2]$ *while the pose of the object* $P_o^{\mathcal{W}}$ *remains unchanged.*

Akin to a robot planning problem, a solution to a motion planning problem is a partial policy $\pi_{mp} : \mathcal{X}_r \to \mathcal{A} : x \mapsto a$ that maps each reachable configuration from the robot configuration space $\mathcal{X}_r$ to a primitive action $a \in \mathcal{A}$. In a deterministic setting, this would translate to a sequence of configurations $\pi_{mp} = [x_i, \ldots, x_g]$ such that for all $x \in \pi_{mp}$, $c(x) = 0$, i.e., every configuration in the motion plan is collision-free. Typically, a sampling-based motion planner, such as RRT (LaValle, 1998) or PRM (Kavraki *et al.*, 1996), are used to solve deterministic problems.

As evident from what we have just discussed, a solution to a motion planning problem only considers the state of a robot. It does not explicitly take various objects and changing configuration spaces into account. Therefore, complex robot planning problems are usually defined as task and motion planning problems.

Task and motion planning problems require a symbolic abstraction of the robot planning problem. Therefore, we start by defining a symbolic abstraction of a robot planning problem and then define a task and motion planning problem.

2.3 Symbolic Abstraction of a Continuous Robot Planning Problem

Symbolic abstractions convert a continuous robot planning problem to a symbolic PDDL (McDermott *et al.*, 1998) problem. A symbolic planning problem is defined using symbolic

predicates (or relations) instead of their continuous counterparts. We define a symbolic predicate (relation) as follows.

Definition 3 *A symbolic predicate (relation)* $p_{\text{sym}}(y_1, \ldots, y_k)$ *is a* ***symbolic relation (function)*** *iff all of its arguments* $y_1, \ldots, y_k$ *are symbolic.*

Let $\mathcal{P}$ be a set of these symbolic predicates (relations). Each predicate $p \in \mathcal{P}$ is parameterized by typed parameters and represents a relation between these objects. Given a set of objects in the environment, a predicate $p \in \mathcal{P}$ can be grounded using the objects in the universe of the model. We refer to a predicate as p and a grounded predicate as p'. Each grounded predicate p' defines a Boolean classifier that evaluates true in a low-level state x (denoted as $p'_x = 1$) if the corresponding relation holds true between the objects used to ground the predicate in the state x.

A symbolic state s is a logical structure or a model defined over the set of symbolic predicates $\mathcal{P}$. Intuitively, abstract states are defined as a set of grounded predicates that evaluate true in a given low-level state. Typically, all abstract states only consist of grounded predicates. However, for the context of this work, we also define a lifted abstract state which is the set of lifted predicates such that at least one grounding of those predicates is true in a given state. We refer to an abstract lifted state as s and an abstract grounded state as s'. Lastly, we define an abstraction function $\alpha : \mathcal{X} \to \mathcal{S}' : x \mapsto s'$ that evaluates every grounded predicate in a low-level state x and returns an abstract state s'.

We define symbolic lifted actions using the set of lifted predicates $\mathcal{P}$. Each abstract action $\bar{a} \in \bar{\mathcal{A}}$ is parameterized with typed symbolic parameters. Each action $\bar{a}$ is defined as a tuple $\langle para_{\bar{a}}, pre_{\bar{a}}, eff_{\bar{a}} \rangle$. Here, $para_{\bar{a}}$ is a list of typed parameters; $pre_{\bar{a}}$ is a conjunctive formula of parameterized predicates from the set of predicates $\mathcal{P}$; eff_a is the effect of the action $\bar{a}$, and it is defined as a tuple $eff_{\bar{a}} = \langle add_{\bar{a}}, del_{\bar{a}} \rangle$ where $add_{\bar{a}}$ is a set of predicates that are added to the state and $del_{\bar{a}}$ is a set of predicates that are removed from the state when

the action $\bar{a}$ is executed. Each predicate in *add*$_{\bar{a}}$ has a probability value associated with it that represents the probability with which the predicate would be added when the action is successfully executed. Similarly to the predicates, an action $\bar{a}$ can be grounded using the objects yielding a grounded action $\bar{a}'$. This also generates grounded precondition *pre*$_{\bar{a}'}$ and grounded effect *eff*$_{\bar{a}'}$. A grounded action $\bar{a}'$ is applicable in a grounded state s' if and only if *pre*$_{\bar{a}'} \models s'$. Formally, every grounded action $\bar{a}' \in \bar{\mathcal{A}}'$ defines a function $\bar{a}' : \mathcal{S}' \rightarrow \mu\mathcal{S}'$ that maps each symbolic state $s' \in \mathcal{S}'$ to a distribution of resultant symbolic states.

We define a symbolic robot planning problem as a stochastic shortest path problem similar to a robot planning problem (Def. 1).

Definition 4 *A **symbolic robot planning problem** is defined as a tuple* $\bar{\mathcal{M}} = \langle \bar{\mathcal{O}}, \mathcal{T}, \mathcal{P}, \bar{\mathcal{A}}, \mathcal{S}', s_i', \mathcal{S}_g' \rangle$ *where,*

- $\bar{\mathcal{O}}$ *is a set of symbolic references to objects and robots in the environment.*
- $\mathcal{T}$ *is a set of object types.*
- $\mathcal{P}$ *is a set of symbolic lifted predicates.*
- $\mathcal{S}'$ *is a set of abstract states.*
- $s_i \in \mathcal{S}'$ *is an initial state.*
- $\mathcal{S}_g' \subset \mathcal{S}'$ *is a set of goal states.*

Example 3 *Fig. 2.2 shows an example of an abstract action defined using symbolic predicates. The precondition specifies that there should exist a valid grasp pose, and a motion plan from the current configuration of the robot to the grasp configuration, and the gripper should be empty. The effect specifies that with probability 1.0, the gripper would be at the grasp configuration, and with probability 0.8, the object would be grasped by the gripper.*

Akin to a low-level robot planning problem, a solution to a symbolic planning problem is a partial policy defined over grounded abstract actions. Formally, a partial policy $\bar{\pi}$:

Pick(obj_1*,* $gripper_1$ $config_1$*,* $config_2$*,* $grasp_pose$*,* $traj_1$*)*

Precondition *RobotAt(*$config_1$*)* , *holding(*obj_1*),*

IsValidMP($traj_1$*,* $config_1$*,* $config_2$*),*

IsCollisionFree($traj_1$*),*

IsEmpty($gripper_1$*),*

IsValidGraspPose(obj_1*,* $config_2$*,* $grasp_pose$*)*

Effect 1.0 ¬ *holding(*obj_1*),*

¬ *RobotAt(*$config_1$*)*, *RobotAt(*$config_2$*),*

at($gripper_1$*,* $grasp_pose$*)*

0.8 ¬ *IsEmpty(*$gripper_1$*),*

Holding($gripper_1$*,* obj_1*),*

Figure 2.2: Specification of an abstract action that picks up an object (obj_1) using a robot ($gripper_1$)

$\mathcal{S}' \rightarrow \bar{\mathcal{A}}'$ maps each reachable abstract state to a grounded symbolic action. Planners such as FF (Hoffmann, 2001), FD (Helmert, 2006), and LAO* (Hansen and Zilberstein, 2001) can be used to compute such solutions.

Symbolic solutions cannot be executed by a robot. They must be converted to a sequence of primitive actions that a robot can execute. Intuitively, a motion plan must be computed for each high-level grounded action. *Pose generators* can be used to convert each high-level action to a motion planning problem. Formally, a pose generator defines an inverse abstraction function. Let γ_p be a pose generator for a lifted symbolic predicate $p \in \mathcal{P}$. For a grounded predicate p', a pose generator $\gamma_{p'} = \{x | x \in \mathcal{X} \wedge p'_x = 1\}$. A pose generator for a grounded state s' is defined as $\bigcap_{p' \in s'} \gamma_{p'}$.

A naíve approach would be to compute a high-level plan first and then use *pose generators* to convert them to a motion plan. However, due to the lossy nature of the abstractions,

such a naíve approach typically does not work. Therefore, task and motion planning approaches such as Cambon *et al.* (2009) and Srivastava *et al.* (2014) interleave high-level planning and refinement for searching a high-level symbolic plan that has motion planning refinements for each action in the high-level plan.

We formally define a task and motion planning problem is defined as follows.

Definition 5 *A task and motion planning problem is defined as a tuple $\langle \mathcal{M}, \alpha, \bar{\mathcal{M}}_\alpha \rangle$ where $\mathcal{M}$ is a robot planning problem, α is an abstraction function, and $\bar{\mathcal{M}}_\alpha$ is a symbolic robot planning problem for the robot planning problem $\mathcal{M}$ obtained using the abstraction function α.*

2.4 Conclusion

This section defines a robot planning problem, a motion planning problem, and a task and motion planning problem. The central focus of this thesis is to learn world models in the form of abstractions that convert the given robot planning problem into a task and motion planning problem. However, before we discuss our approach to learning such abstractions, we must discuss approaches that use symbolic world models for task and motion planning.

Chapter 3

STOCHASTIC TASK AND MOTION PLANNING

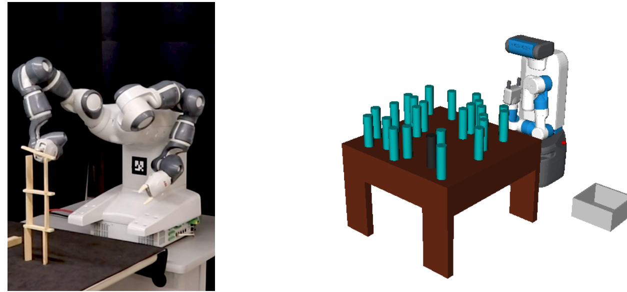


Figure 3.1: Left: YuMi robot uses HPlan (Sec. 3.1) to build a 3π structure (2 π structures stacked on top of a π structure) with Keva planks despite uncertainty in their initial locations. Right: A stochastic variant of the cluttered table domain where the robot is instructed to pick up the black can, but pickups may fail and crush the cans requiring them to be disposed of.

A long-standing goal in robotics is to develop robots that can operate autonomously in real-world environments and solve complex tasks such as cleaning a room or organizing a table. This chapter presents an approach for using a hand-provided *entity abstractions* – a novel abstraction method for abstracting robot planning problems – to compute solutions for such complex problems in stochastic settings.

We develop a hierarchical anytime approach that uses symbolic abstractions in the form of abstract predicates and actions (as discussed in Sec. 2.3) for a robot planning problem (Def. 1) and computes a refined task and motion policy. Our anytime algorithm (Sec. 3.1) is guaranteed to find a solution, if it exists, as well as ensure a quick solution that the robot can begin to execute without waiting for a full policy computation.

This chapter first explains the nature of input abstractions (Sec. 3.1). Then, it describes our algorithm – HPlan – for computing task and motion policy (Sec. 3.1) for a task and motion planning problem (Def. 5), and lastly, it provides a strong empirical evaluation of

our approach in several settings where complex planning solutions are required to solve the task (Sec. 3.3).

3.1 Entity Abstraction

We define a specific abstraction (*entity abstraction*) as the abstraction function α to provide interpretations for symbolic predicates as discussed in Sec. 2.3. Let $\mathcal{O}_l$ $(\mathcal{O}_h)$ be the universe for the first-order logic vocabulary $\mathcal{V}_l$ $(\mathcal{V}_h)$ such that $|\mathcal{O}_h| \leq |\mathcal{O}_l|$. Here $\mathcal{O}_l$ and $\mathcal{V}_l$ are low-level concrete universes and vocabularies respectively and $\mathcal{O}_h$ and $\mathcal{V}_h$ are their abstract counterparts. Let $\rho : \mathcal{U}_h \rightarrow 2^{\mathcal{U}_l}$ be a collection function that maps elements in $\mathcal{U}_h$ to the collection of $\mathcal{U}_l$ elements that they represent, e.g., $\rho(Table) = \{loc| \wedge_i \; loc \cdot BoundaryVector_i < 0\}$. Here ρ binds the symbolic reference `Table` $\in \mathcal{U}_h$ to a set of locations in $\mathcal{U}_l$ that are enclosed by some polygonal boundary.

We define entity abstraction α_ρ using the collection function ρ as $[\![r]\!]_{\alpha_\rho(V_l)}(\tilde{o_1}, \ldots, \tilde{o_n}) = True$ iff $\exists\, o_1, \ldots, o_n$ such that $o_i \in \rho(\tilde{o_i})$ and $[\![\psi_r^{\alpha_\rho}(o_1, \ldots, o_n)]\!]_{S_l} = True$. Here, r and ψ_r are formulas defined over vocabularies $\mathcal{V}_l$ and $\mathcal{V}_h$ respectively. We omit the subscript ρ when it is clear from the context. Entity abstractions define the truth values of predicates over abstracted entities as a disjunction of the corresponding concrete predicate instantiations. E.g., an object is in the abstract region "kitchen" if it is at one of any locations in that region and an object is on "table" if it is at any location on the table-top. Such abstractions have been used for efficient generalized planning (Srivastava *et al.*, 2008) as well as answer set programming (Saribatur *et al.*, 2019). These types of abstractions introduce terms that may not be recognizable (or evaluable) at the high level which makes these abstractions lossy and high-level models obtained by these abstractions inaccurate. E.g., the exact location of the table or the trajectory used to reach a configuration from the current configuration.

We define a *pose generator* for each predicate using the collection function of the argu-

Algorithm 1: HPlan Algorithm

Input: model $\mathcal{M}$, abstraction function α, concretization function γ, abstract model $\bar{\mathcal{M}}_\alpha$, symbolic planner P

Output: anytime, contingent policy that is executable in $\mathcal{M}$

1. Initialize PRG with a node with an abstract policy $\bar{\pi}$ for $\mathcal{G}$ computed using *P*;
2. **while** *solution of desired quality not found* **do**
3. $\quad u$ ← GetPRNode();
4. $\quad \bar{\mathcal{M}}_u$ ← GetAbstractModel(u);
5. $\quad \bar{\pi}_u$ ← GetAbstractPolicy($\bar{\mathcal{M}}_u$, $\mathcal{G}$, P, u);
6. $\quad$ Choice ← NDChoice$\{$*RefinePolicy*, *RefineAbstraction*$\}$;
7. $\quad$ **if** *Choice* = RefinePolicy **then**
8. $\quad\quad$ **while** $\bar{\pi}_u$ *has an unrefined RTL path and resource limit is not reached* **do**
9. $\quad\quad\quad path$ ← GetUnrefinedRTLPath($\bar{\pi}_u$);
10. $\quad\quad\quad$ **if** *explore*`// non-deterministic`
11. $\quad\quad\quad$ **then**
12. $\quad\quad\quad\quad$ replace a suffix of refined partial $path$ with a random action;
13. $\quad\quad\quad$ Search for a feasible concretization of $path$;
14. $\quad$ **if** *Choice* = RefineAbstraction **then**
15. $\quad\quad path$ ← GetUnrefinedRTLPath($\bar{\pi}_u$);
16. $\quad\quad \sigma$ ← ConcretizeFirstUnrefinedAction($path$);
17. $\quad\quad$ failure_reason ← GetFailedPrecondition(σ);
18. $\quad\quad \bar{\mathcal{M}}'$ ← UpdateAbstraction($\bar{\mathcal{M}}$, failure_reason) ;
19. $\quad\quad \bar{\pi}'$ ← merge($\bar{\pi}$, GetAbstractPolicy($\bar{\mathcal{M}}'$, $\mathcal{G}$, solver));
20. $\quad\quad$ generate_new_pr_node($\bar{\pi}'$, $\bar{\mathcal{M}}'$);
21. $\quad$ recompute p/c ratio for unrefined RTL paths;

ments of the predicate. E.g., a pose generator for a predicate `On(A, Table)` with a symbolic reference `Table` with a collection function $\rho(Table) = \{loc \mid \wedge_i\ loc{\cdot}BoundaryVector_i < 0\}$, the pose generator is defined as $\Gamma = \{loc | loc \in \rho(Table)\}$.

3.2 Computing Task and Motion Policies

We extend the idea of planning with abstractions briefly discussed by Srivastava *et al.* (2016b) to perform task and motion planning in stochastic environments using abstraction hierarchies. The goal is to find a valid high-level policy that also has valid low-level refinements for each of its actions. We propose the HPlan algorithm (Alg. 1) that performs hierarchical planning with arbitrary abstraction and concretization functions.

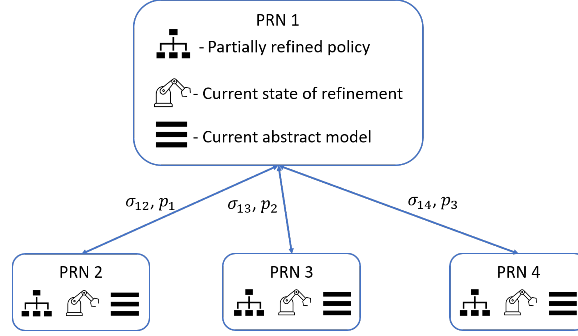


Figure 3.2: Plan refinement graph (PRG) used to maintain separate abstract models. Each plan refinement node (PRN) contains an abstract model, partially refined policy, and current state of refinement. Each edge contains refinement for a partial policy (σ_{ij}) and a failure reason (p_k).

HPlan (Alg. 1) uses a policy refinement graph (PRG) to keep track of different abstract models and their corresponding policies. As shown in Fig. 3.2, each node u in a PRG contains an abstract model $\bar{\mathcal{M}}_u$, an abstract policy $\bar{\pi}_u$, and the current state of refinement for each action $\bar{a}_j \in \bar{\pi}_u$. An edge (u, v) in a PRG from a node u to a node v consists of a partial refinement of the policy (σ_{uv}) and a failed precondition of the first action from $\bar{\pi}_u$ that lacks valid motion planning refinement. Our approach combines two processes: $1)$ concretizing the abstract policy, and $2)$ refining the abstract model.

HPlan (Alg. 1) interleaves these two steps. The algorithm starts by initializing the PRG with a node containing this abstract model $\bar{\mathcal{M}}$, and an abstract policy $\bar{\pi}$ computed using an off-the-shelf symbolic solver that achieves the goal $\mathcal{G}$ (line 1). Each iteration of the main loop (line 2) selects a policy refinement node (PRN) u from the PRG using a defined strategy (line 3). Arbitrary strategies can be used to make this selection. HPlan uses an off-the-shelf task planner to compute a high-level policy for the current abstract model if the selected PRN does not already have a high-level policy (line 5). Once a policy is computed (or obtained), HPlan non-deterministically decides (line 6) to either refine the high-level policy in the selected PRN by instantiating abstract arguments of actions in the policy (lines 7-13) or to update the high-level abstractions to compute accurate high-level

policies (lines 14-20). The algorithm carries out these interleaved steps as follows:

a) Concretizing the abstract policy Lines *8-13* search for a valid concretization (refinement) of the high-level policy selected/computed on line 5 by concretizing the abstract actions with actions from the concrete domain $\mathcal{M}$ using the concretization function Γ_α as explained in Sec. 3.1. To refine a high-level policy, a root-to-leaf (RTL) path is selected that has at least one unrefined action. Each unrefined action is concretized using a local backtracking search (line 13) (Srivastava *et al.*, 2014). A concretization $c_0, a_1, c_1, \ldots, a_k, c_k$ is a valid concretization of an RTL path $\bar{s}_0, \bar{a}_1, \bar{s}_1, \ldots, \bar{a}_k, \bar{s}_k$ is valid iff $c_{i+1} \in a_{i+1}(c_i)$ and $c_i \vDash precon(a_i + 1)$ for $i = 0, \ldots, k-1$. A policy is refined when concretization for each action in every RTL path in the policy is computed. However, due to the lossy nature of the abstraction, it may be that no valid concretization exists for the policy $\bar{\pi}_u$. For example, consider the `Pick` action shown in Fig. 2.2, and consider an abstraction which drops *In-Collision* predicate that checks whether a trajectory is in collision with some object while trying to pick the object. Such high-level actions would not have any valid concretization if all the trajectories were being obstructed by some object at the low level.

b) Refining the abstract model Lines *15-20* fix a concretization for the partially refined policy selected on line 5 and identify the earliest abstract state in the selected policy whose subsequent action's concretization is infeasible. The abstract model is refined by adding the true form of the violated precondition at the low level. Continuing the same example, if all the trajectories from the current state to the state with gripper at the grasp pose of the object are in collision with some object, the concrete precondition *InCollision(traj,* obj_x*)* is violated at the concrete level and is added to the current abstract model. The rest of the policy after this abstract state is discarded. Lines *19-20* use the new model to compute a new policy. The symbolic planner is invoked to compute a new policy from the updated

state; its solution policy is unrolled as a tree of bounded depth and appended to the partially refined path. This allows the time horizon of the policy to be increased dynamically.

Theorem 3.2.1 *If there exists a proper policy – the probability of reaching the goal is* 1.0 *– that reaches the goal within horizon* h*, and has feasible low-level concretization for each of its actions, and the probability measure of these refinements under the probability density of the pose generators is non-zero, then Alg. 1 will find it with probability* 1.0 *in the limit of infinite samples.*

Proof 3.2.1 *Let* π_p *be the proper policy that achieves the goal with horizon* h *and has valid low-level concretization for each of its actions. Consider a policy* π_i *inside a PRN* i *at an intermediate step of Alg. 1; let* k *denote the minimum depth up to which* π_p *and* π_i *match. Here,* k *denotes a measure of correctness. When PRN* i *is selected for refinement, eventually Alg. 1 would try to compute low-level concretization for an action at depth* $k+1$ *that does not match with the proper policy* π_p*. In this case, there is a chance that Alg. 1 would select the correct action (that matches with* π_p *at depth* $k+1$*) under the* explore *condition (lines* 10-12*) of Alg. 1 and then generates a plan that reaches the goal state. A finite number of discrete actions in the abstract model and the fixed horizon ensures that in time bounded in expectation, HPLan will generate a policy with the measure of correctness* $k+1$ *and eventually with the measure of correctness* h*. Once the algorithm finds the policy with the measure of correctness* h*, it stores it in the PRG and is guaranteed to find feasible refinements with probability one if the measure of these refinements under the probability density of the generators is non-zero.*

3.2.1 HPlan for STAMP

We enhance the basic Alg. 1 in two primary directions to facilitate stochastic task and motion planning (STAMP) problems. These optimizations allow Alg. 1 to compute any-

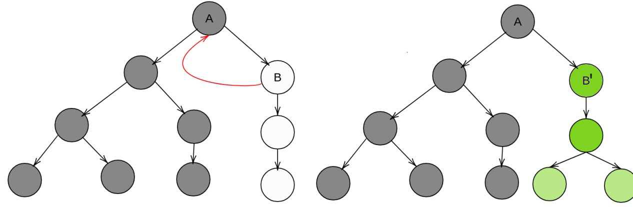


Figure 3.3: Left: Backtracking from node B invalidates the concretization of subtree rooted at A. Right: Replanning from node B

time solutions for STAMP problems and improve the search of concretization of abstract policies.

Search for concretizations Sampling-based backtracking search performed by Alg. 1 (line 13) to concretize the abstract actions suffers from a few limitations in stochastic settings that are not present in the deterministic settings. Fig. 3.3 illustrates the problem. The gray nodes in the image show the actions which are concretized. White nodes represent actions that are yet to be concretized. Sibling nodes represent the non-deterministic action outcomes. Now, if the action in node B does not accept any valid concretization, backtracking to node A and changing its action's concretization would invalidate concretization for the entire subtree rooted at node A. Alg. 1 handles such scenarios by non-deterministically selecting whether to perform backtracking searching or not (line 6) and by maintaining different abstract models through *PRG* and employing a resource limit (line 8) to explore them simultaneously.

Anytime computation for task and motion policies The main computational challenge for Alg. 1 in stochastic settings is that the number of root-to-leaf (RTL) branches grows exponentially with the time horizon and the number of contingencies in the domain. In most scenarios, not all contingencies are equally probable. Each RTL path has a certain probability of being encountered; refining it incurs a computational cost. Waiting for a complete

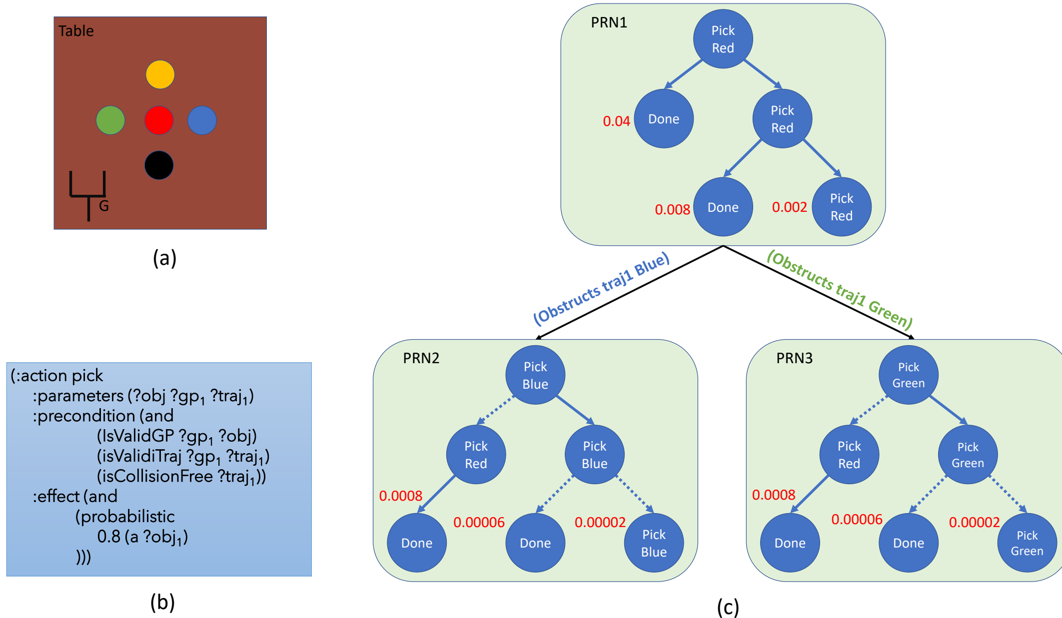


Figure 3.4: A working example for Alg. 1. (a) shows the initial environment configuration. The goal for the robot is to pick up the "Red" object which is surrounded by "Blue", "Green", "Orange", and "Black" objects. G is the end-effector of a robot. (b) shows a high-level, abstract task specification of the "pick" action. (c) shows the policy refinement graph (PRG) which is generated incrementally by Alg. 1. Each green box represents a policy refinement node (PRN). The tree in each PRN represents a high-level policy. Each node in a high-level policy is a state-action pair. For brevity, we only show high-level action in the node. Trees with dotted lines are partial policies. The red number represents p/c ratio for each RTL path in a policy.

refinement of the policy tree wastes a lot of time as most of the situations have a very low probability of being encountered. The optimal selection of the paths to refine within a fixed computational budget can be reduced to the knapsack problem. Unfortunately, we do not know the precise computational costs required to refine an RTL path. However, we can approximate this cost depending on the number of actions in an RTL path and the size of the domains of the arguments of those actions. Furthermore, the knapsack problem is NP-hard. Luckily, we can compute provably good approximate solutions to this problem using a greedy approach: we prioritize the selection of a path to refine based on the probability of encountering that path p and the estimated cost of refining that path c. We compute p/c ratio for all the paths and select the unrefined path with the largest ratio for refinement (line

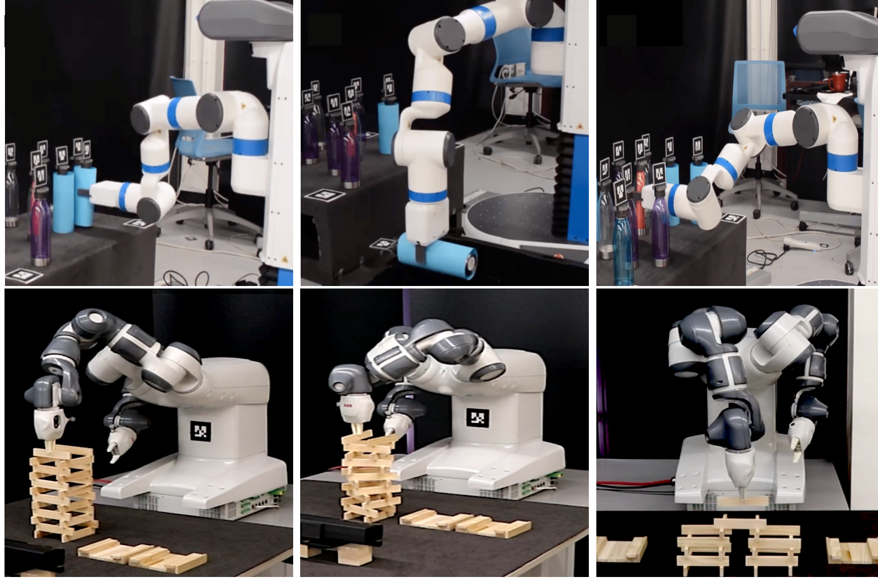


Figure 3.5: Top: Cluttered Table: The Fetch mobile manipulator uses a STAMP policy to pick up a target bottle while avoiding those likely to be crushed. It replaces a bottle that was not crushed (left), discards a bottle that was crushed (center) and picks up the target bottle (right). Bottom: Building Structures with Keva Planks: ABB YuMi builds Keva structures using a STAMP policy: 12-level tower (left), twisted 12-level tower (center), and 3-towers (right).

9 and 15). The p/c ratio for each path is updated after each iteration of the main loop (line 21). Intuitively, our approach works as follows:

Example Fig. 3.4 illustrates our approach for solving a STAMP problem using Alg. 1. Fig. 3.4(a) shows a low-level configuration of an environment. Here, a robot with an effector G is asked to pick up the red object which is surrounded by green, blue, orange, and black objects. Fig. 3.4(b) shows a high-level specification of the pick action in the PPDDL format. Fig. 3.4(c) shows the policy refinement graph (PRG) that is generated incrementally by Alg. 1.

As explained earlier in the section, Alg. 1 starts with a single node in the PRG – in this case, PRN1. Initially, PRN1 does not have a high-level policy. Alg. 1 uses the abstract action descriptions (abstract model $\bar{\mathcal{M}}$) and an off-the-shelf high-level SSP solver to compute a high-level symbolic policy that reaches the abstract goal (line 5) and computes p/c ratios

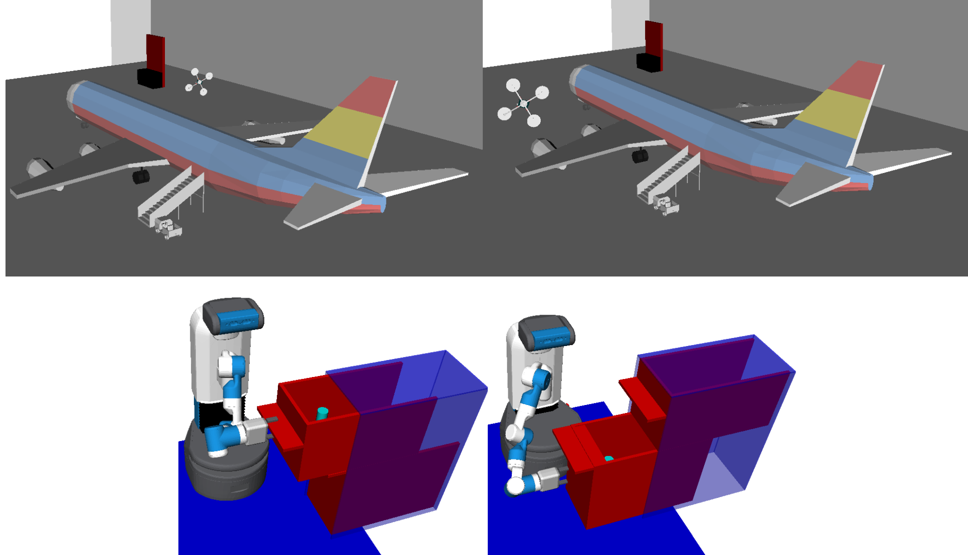


Figure 3.6: Top: Aircraft Inspection: UAV inspects faulty parts of an aircraft in an airplane hangar and alerts the human about the location of the fault. UAV's movements and sensors are noisy, so it may drift from its location or fail to locate the fault. Bottom: Find the can: Fetch searches for a can in drawers. The can can be placed in one of the drawers stochastically.

for each RTL path in this abstract policy. To compute this ratio, we estimate the cost of refining each high-level action as follows: Suppose that the generators used to concretize the pick actions samples four grasp poses in four cardinal directions to pick up the object and five motion planning trajectories between the robot's current configuration to the grasp pose, then the approximate cost of refining this action would $4 \times 5 = 20$. We use this approximate cost to compute p/c ratios (red numbers in Fig. 3.4). The next step for Alg. 1 is to non-deterministically decide between refining the computed high-level policy and refining the abstraction. Assume Alg. 1 non-deterministically decides to refine the high-level policy (line 6). After deciding to refine the high-level policy, Alg. 1 selects an RTL path using the p/c ratio and tries to refine each action on this path by instantiating each symbolic argument. Here in this example, the first RTL path would only have a single high-level action *pick(Red,* gp_1*,* $traj_1$*)* that needs refinement. To instantiate the high-level pick action, it first uses a generator to sample one of the possible grasp poses for the red object and then uses

a low-level motion planner to generate a trajectory that would take the robot end-effector G to the selected grasp pose from its current pose. As the red object is surrounded by other objects, all the trajectories that take the end-effector to the grasp pose, are in collision with at least one object. This violates the precondition of the pick action making the refinement infeasible. Alg 1 continues trying to refine this action using the local and global backtracking search for a fixed amount of time before again making a non-deterministic choice between refining the high-level policy or the high-level abstraction.

Suppose this time Alg. 1 decides to refine the high-level abstraction. To do so, it would identify the failing precondition preventing a valid refinement for the high-level policy and generate a set of child nodes in the PRG – PRN2 and PRN3 in this case corresponding to failing preconditions *Obstructs(traj*$_1$*, Blue)* and *Obstructs(traj*$_1$*, Green)*. Once these nodes are generated, Alg. 1 would move on to the next iteration of the approach where it would select one of these newly generated plan refinement nodes and repeat the entire process until a complete task and motion policy is computed.

Theorem 3.2.2 *Let t be the time since the start of the algorithm at which the refinement of any* RTL *path is completed. If path costs are accurate and constant then the total probability of unrefined paths at time t is at most $1 - opt(t)/2$, where $opt(t)$ is the best possible refinement (in terms of the probability of outcomes covered) that could have been achieved in time t.*

Proof 3.2.2 *(*Sketch*) The proof follows from the fact that the greedy algorithm achieves a 2-approximation for the knapsack problem. In practice, we estimate the cost as $\hat{c}$, the product of measures of the true domains of each symbolic argument in the given* RTL*. Since, $\hat{c} \geq c$ modulo constant factors, the priority queue never can only underestimate the relative value of refining a path, and the algorithm's coverage of high-probability contingencies will be closer to optimal than the bound suggested in the theorem above. This optimization*

gives a user the option of starting execution when the desired value of the probability of covered contingencies has been reached.

3.3 Empirical Evaluation

We use a total of five domains with varying configurations to evaluate our approach. All these five domains had a mix of deterministic and stochastic actions. We use an implementation of LAO* (Hansen and Zilberstein, 2001) from the MDP-Lib (Pineda, 2014) repository for computing policies for SSPs. We use OpenRAVE (Diankov, 2010) robot simulation system with its collision checkers to represent 3D environments and perform collision checking. We also use CBiRRT's (Berenson *et al.*, 2009) implementation from the PrPy (Koval, 2015) suite for computing motion plans. In practice, fixing the horizon H for the SSP solver apriori is infeasible and renders some problems unsolvable. Instead, we implemented a variant that dynamically increases the horizon until the goal is reached with a probability $p > 0$. [1]

Lagriffoul *et al.* (2018) propose several framework-independent benchmark domains for task and motion planning systems. While these benchmarks are proposed for deterministic TAMP systems, characteristics of the domains can still be used to evaluate STAMP systems. Table 3.1 shows the criteria fulfilled by every domain used to evaluate our approach. We include the average number of branches in the policy tree as an additional criterion to depict the complexity of stochastic problems.

Problem 1: cluttered table In this problem, we have a table cluttered with cans, each having different probabilities of being crushed when grasped by the robot incurring a high cost (probability for crushing was set to 0.1, 0.5 & 0.9 in different experiments in Fig. 3.8(a)), while others are normal cans that cannot be crushed. The goal of the robot is

[1] The source code of the framework along with the videos of our experiments can be found at `https://aair-lab.github.io/stamp.html`

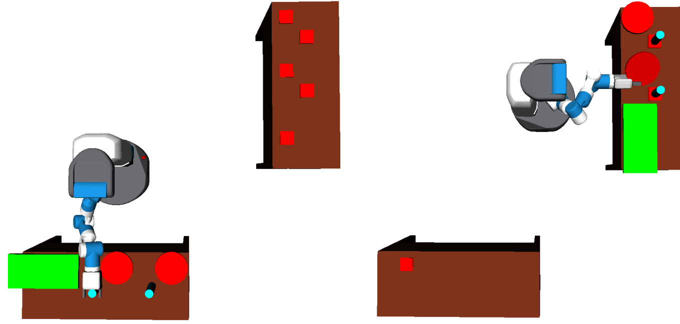


Figure 3.7: Setting up a dining table: Fetch uses STAMP policy to set up a dining table. A tray is available to carry multiple items at a time but carrying more than two items on the tray may break the items. Left: The initial state. Right: The goal state.

to pick up a specified can. We used different numbers of cans (15, 20, 25) and different random configurations of cans to extensively evaluate the proposed framework. We also used this scenario to evaluate our approach in the real world (Fig. 3.5) using the Fetch robot Wise *et al.* (2016).

Problem 2: aircraft inspection In this problem, an unmanned aerial vehicle (UAV) is employed to inspect possibly faulty parts of an aircraft in an airplane hangar. The goal for the agent is to locate the fault and notify the human supervisor about it. Fig. 3.6 shows the simulated environment. The UAV's sensors are inaccurate and may fail to locate the fault with some non-zero probability (failure probability was set to 0.05, 0.1, and 0.15 for experiments in Fig. 3.8) while inspecting the location; it may also drift to another location while flying from one location to another or while inspecting the parts. The UAV has a limited amount of battery charge. A charging station is available for the UAV to dock and charge itself. All movements use some amount of battery charge depending on the length of the trajectory, but the high-level planner cannot determine whether the current level of charge is sufficient for the action or not as it lacks details such as current battery level, length of previous and next trajectories, etc. This makes it necessary to have an interleaved approach that searches for a high-level policy that has valid low-level refinements.

Criteria	Cluttered Table	Aircraft Inspection	Building Keva Structures	Kitchen	Find the can
Infeasible Tasks	✓	✓			
Large task spaces	✓	✓	✓	✓	
Motion/task trade-off	✓	✓	✓	✓	
Non-monotonicity	✓			✓	✓
#branches	$O(2d)$	$O(4^h)$	$O(2^n)$	2	2

Table 3.1: Critera defined by Lagriffoul *et al.* (2018) evaluated in each of the test domains.

Problem 3: building structures with keva planks In this problem, the YuMi robot (ABB, 2015) is used to build different structures using Keva planks. Keva planks are laser-cut wooden planks with uniform geometry. Fig. 3.5 and Fig. 3.1 show the target structures. Planks are placed one at a time by a user after each pickup and placement by the YuMi. Each new plank may be placed at one of a few predefined locations, which adds uncertainty to the planks' initial location. For our experiments, two predefined locations were used to place the planks with a probability of 0.8 for the first location and a probability of 0.2 for the second location. In this problem, handwritten goal conditions are used to specify the desired target structure. The YuMi needs a task and motion policy for successively picking up and placing planks to build the structure. There are infinitely many configurations in which one plank can be placed on another, but the abstract model blurs out different regions on the plank. The generator that samples put-down poses for planks on the table uses the target structure to concretize each plank's target put-down pose. The number of branches in a solution tree grows exponentially with the number of planks in the structure and can quickly become huge. For example, a solution tree for a structure with just 10 planks would have a total of 1024 branches. Due to the large state space, the state-of-the-art SSP solver used for other domains failed to compute a high-level policy for these problems. Our observation shows that most SSP solvers fail to compute a high-level solution for structures that have greater than 6 planks. However, these structure-building problems exhibit repeating substructures every 1-2 layers that reuse minor variants of the same abstract policy. We

used this observation and used a generalized SSP solver (Karia *et al.*, 2022) that computes generalized policies for SSPs with such repeating patterns. Other approaches for generalized planning (Srivastava *et al.*, 2008; Bonet *et al.*, 2009; Hu and De Giacomo, 2011; Srivastava *et al.*, 2011) can also be used to automatically extract and utilize such patterns in other problems with repeating structures.

Problem 4: setting up a dining table In this problem, the Fetch robot arranges a dining table with two plates and two glasses (Fig. 3.7). A tray is available for the robot to use for carrying multiple items at once. If the robot tries to carry more than two objects on a tray at once, the objects can fall from the tray with a probability of 0.2 and that would break the objects. While using the tray can reduce the number of trips between tables, breaking the objects would render the problem unsolvable. As our approach considers all possible outcomes of stochastic actions, it successfully computes a policy that prevents any object from breaking compared to determinization-based approaches that only consider the most likely outcome for stochastic actions that may fail to solve such problems as most-likely scenarios might fail to capture dead ends in the domain.

Problem 5: find the can In this problem, the *Fetch* robot searches for a can that may be present in one of the drawers. Fig. 3.6 shows the simulated environment for the problem. The can is placed in one of the drawers with a given prior distribution. The robot does not have access to the can's location apriori and has to open the drawer to check whether the can is present in the drawer or not. In our experiments, the can is placed in the upper drawer with a probability of 0.6 and in the bottom drawer with a probability of 0.4.

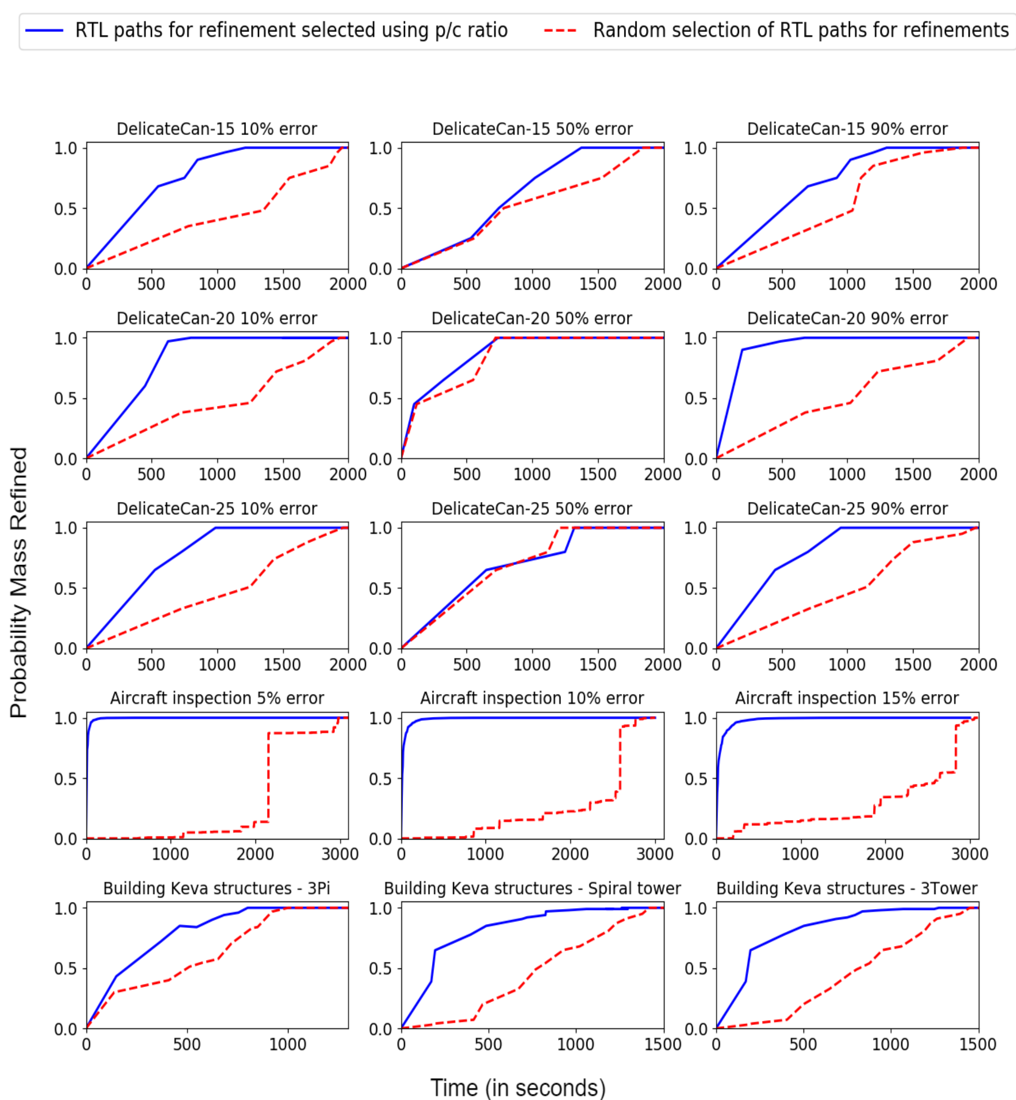


Figure 3.8: Anytime performance of ATM-MDP, showing the time in seconds (x-axis) vs. probability mass refined (y-axis).

3.3.1 Analysis of the results

Nature of the solutions The most distinct characteristic of the solutions generated through our framework is that they capture all possible contingencies that may arise while executing the policy. E.g., solutions generated for setting up the dinner table (problem 4) avoid placing more than two items on the tray to eliminate the possibility of incurring higher

Problem	% Solved	Avg. Time (s)
Cluttered-15	100	1120.21 ± 1014.54
Cluttered-20	83	1244.32 ± 990.65
Cluttered-25	75	1684.54 ± 890.78
Aircraft Inspection	100	2875.01 ± 103.65
3π	100	1356.34 ± 75.8
Tower-12	100	2232.36 ± 104.84
Twisted-Tower-12	80	3249.92 ± 773.69
Setting up a dining table	100	1287.23 ± 321.32
Find the can	100	36.74 ± 0.13

Table 3.2: Summary of times taken to solve the STAMP problems. Timeout: 4000 seconds.

expected cost, and solutions for picking up a can from the cluttered table (problem 1) avoid picking up a delicate can for similar reasons.

Quality of the solutions over time While our approach computes refinements for every action in the policy, the anytime property allows the agent to start executing the actions before all the actions are refined. Our approach computes anytime policies with respect to the possible outcomes handled by a policy at any point in time. Fig. 3.8 shows the anytime property of our approach in stochastic test domains. The y-axis shows the probability with which the policy available at any point of time during the algorithm's computation will be able to handle all possible outcomes, and the x-axis shows the time (in seconds) required to compute task and motion policies that handle these outcomes. The results show that with time, the likelihood with which the solution would be able to handle any scenario increases. The agent can use this observation to decide a threshold at which it can start executing the actions. For our experiments, we use a threshold of 60% of all possible outcomes to start the execution of the policy. Our experiments show that in most cases, the problem was solved significantly faster compared to starting execution after refining the entire policy tree (Fig. 3.2).

Impact of prioritized *RTL* path selection The results presented in Fig. 3.8 indicate that when *RTL* paths are selected using the p/c ration (blue line), the framework can quickly

handle outcomes with most likely outcomes, compared to a randomized selection of *RTL* paths for refinements (red line). In most cases, 80% of probable executions are covered within about 30% of the total computation time. This characteristic is most evident in the *aircraft inspection* problem due to a large number of possible outcomes and differences in the probability of different outcomes. Such a prioritization does not make a significant impact if all the outcomes are equally probable. E.g., such impact is the least evident in the *cluttered table* problem with the probability of crushing the objects set to 0.5 given each outcome becomes equally probable and the sequence in which they are handled does not make any difference.

Scalability of the framework Fig. 3.2 shows the time taken by our approach to compute complete STAMP solutions by concretizing every action in the entire policy for the given test problems respectively. We combine results for different variants of the test problem as variations in the probabilities of outcomes do not affect the time required to concretize all actions in the entire policy. Values in Fig. 3.2 are averages of 50 runs with standard deviation. Our empirical evaluation shows that solving a STAMP problem requires significantly more time than an equivalent TAMP problem. E.g., the stochastic variant of the aircraft inspection problem takes nearly 15 times more time than the deterministic version as the stochastic variant had 780 branches in the solution tree compared to a single branch in the deterministic variant. These results reinforce our hypothesis that an anytime approach that prioritizes high-probability scenarios over low-probability situations but still considers all possible outcomes suits better than an approach that does not consider all possible outcomes while showing the scalability of our approach to solve large problems. Results for larger problems such as *Twisted-Tower-12* and *Cluttered-25* show the *scalability* of our system. Even though our approach needs a significant time to compute solutions for such huge problems due to a large number of RTL paths in the policy trees, it was able to solve

almost all problems in these problem settings.

Now, we present a discussion on related approaches.

3.4 Related Work

Stochastic task and motion planning combines hierarchical planning, task planning, planning under uncertainty, motion planning, and task and motion planning together. In this section, we discuss some of the related approaches in these areas of research.

3.4.1 Stochastic Task Planning

Many approaches have been developed for classical planning efficiently in recent years (Blum and Furst, 1997; Bonet and Geffner, 2001; Hoffmann, 2001). Similarly, numerous approaches have been developed to solve stochastic shortest-path problems. Dynamic programming algorithms such as value iteration and policy iteration can be used to compute policies for SSPs. *Real-time dynamic programming* (RTDP) (Barto *et al.*, 1993) generalizes *Korf's Learning-Real-Time-A** algorithm to a trial-based dynamic programming method that ignores a large part of the state-space by only expanding states encountered in trials to solve $SSPs$ faster. *LAO** (Hansen and Zilberstein, 2001) uses heuristics to expand the partial policy tree along with local value iteration to compute policies for SSPs. *Labeled RTDP* (Bonet and Geffner, 2003) extends *RTDP* by labeling states that have converged greedy policy to reduce the number of states considered for expansion to decrease policy computation time. Muise *et al.* (2012) use state relevance to guide the search to reduce the time to compute the policy. Abdelhadi and Cherki (2019) provide a method that decomposes an SSP into multiple smaller $SSPs$ and combines the solution to handle *dead ends*.

3.4.2 Hierarchical Planning

Hierarchical approaches (Sacerdoti, 1974; Knoblock, 1990; Erol *et al.*, 1995; Seipp and Helmert, 2018) use abstractions to generate different hierarchies of relaxed planning problems to compute a solution for a complex planning problem. State abstraction generates hierarchies by removing certain predicates (in relational domains) or variables (in factored domains) from the domain vocabulary. ABSTRIPS (Sacerdoti, 1974) is one of the earliest hierarchical planning approaches which assigns a rank to each literal using a predefined order and the complexity of achieving that literal in the STRIPS planning process. Abstraction hierarchy is generated by dropping literals from the precondition of actions in the domain in the order specified by the rank of literals. The planning hierarchy generated using ABSTRIPS is common for all problems in the given domain and it is not tailored to independent problems.

ALPINE (Knoblock, 1990) uses *ordered monotonicity* to overcome this issue by generating abstraction hierarchies tailored to each problem for the given domain. Seipp and Helmert (2013, 2018) use counter-example guided abstraction refinement (CEGAR) to solve a complex planning problem hierarchically using Cartesian abstraction – a variant of predicate abstraction. This CEGAR-based approach starts with a naïve abstraction for the problem and computes an optimal plan for the abstract model. It tries to execute this plan in the original model. If it fails to execute the plan successfully, it computes a flaw in the current plan and uses it to refine the current abstract model. This approach requires a pre-image of each *grounded operator* and a bounded branching factor for the search tree. Such approaches are not conducive to task and motion planning setups because they require discrete action and state spaces while task and motion planning operates in continuous states and action spaces.

Temporal abstractions generate high-level actions that are compositions of multiple

low-level actions. Some hierarchical planning approaches employ temporal abstraction to create relaxed problems. Multiple approaches Kambhampati *et al.* (1998); Bacchus and Kabanza (2000); Bercher *et al.* (2014) have used hierarchical task networks (HTNs) (Erol *et al.*, 1995) to compute plans efficiently for complex tasks. HTNs use temporal abstractions to define tasks over primitive actions. The goal is to compute a final plan which is a composition of the high-level tasks that are achieved through the partial order planning of the primitive actions. Marthi *et al.* (2007b) compute hierarchical domain descriptions based on angelic semantics using temporal abstractions. They use a top-down forward search algorithm to refine the high-level actions into a sequence of primitive actions. While this approach and HTN-based approaches efficiently perform top-down planning using temporal abstraction, they fail to compute accurate plans in the models that do not fulfill the downward refinement property. Additionally, they do not handle stochasticity.

Several approaches utilize abstraction for solving MDPs (Gopalan *et al.*, 2017; Hostetler *et al.*, 2014; Bai *et al.*, 2016; Li *et al.*, 2006; Singh *et al.*, 1995). However, these approaches assume that the full, unabstracted MDP can be efficiently expressed as a discrete MDP. Marecki *et al.* (2006) consider continuous-time MDPs with finite sets of states and actions. In contrast, our focus is on MDPs with high-dimensional and uncountable state and action spaces. Recent work on deep reinforcement learning (e.g., (Hausknecht and Stone, 2016; Mnih *et al.*, 2015)) presents approaches for using deep neural networks in conjunction with reinforcement learning to solve short-horizon MDPs with continuous state spaces. These approaches can be used as primitives in a complementary fashion with task and motion planning algorithms, as illustrated in recent promising work by Wang *et al.* (2018).

Task planning efficiently computes solutions for complex goals. However, it can not handle manipulation problems with continuous domains that have an infinite branching factor. Though *PDDL 2.1* (Fox and Long, 2003) allows using continuous variables, it still struggles to handle an infinite branching factor.

3.4.3 Motion Planning

Recent research resulted in significant improvements in sampling-based motion planners. Probabilistic roadmaps (PRM) (Kavraki *et al.*, 1996) randomly sample from the C-space to generate a roadmap that can be lazily used to generate motion plans. Rapidly-exploring random trees (RRT) (Lavalle, 1998) computes a collision-free path from an initial robot configuration to the target configuration by connecting randomly sampled robot configurations from the C-space. Bidirectional RRT (BiRRT) (Kuffner and LaValle, 2000) updates existing RRT to initiate search trees from the initial and goal configurations to boost the speed of motion planning. Constrained BiRRT (CBiRRT) (Berenson *et al.*, 2009) extends the BiRRT technique constraining the search space by using projection techniques to explore configurations spaces and finds bridges between them.

3.4.4 Integrated Task and Motion Planning

Most of the prior work in the field of integrated task and motion planning has focused on solving deterministic task and motion planning problems. Most of these approaches can be classified into three categories: *1)* approaches that use symbols to guide the low-level motion planning, *2)* approaches that extend high-level representations to simultaneously search high-level plans along with continuous parameters, and *3)* approaches that use interleaved search for valid high-level plans with low-level refinements for its actions. Our approach falls under the last category. Garrett *et al.* (2021) present an exhaustive survey of these approaches; we discuss only the most closely related approaches here.

Approaches that use symbols to guide the motion planning: Cambon *et al.* (2009) introduced one of the earliest approaches named aSyMov. ASyMov uses symbolic knowledge to guide planning in geometric space using location references. Plaku and Hager (2010) use a similar approach to allow combined task and motion planning for robots with

constrained manipulators. Such approaches employ task planning as a heuristic for planning in the C-space, which may not always be efficient due to a lack of knowledge of geometric constraints at the task-planning level. To overcome this limitation, we interleave the process of computing motion plans and updating the high-level specification.

Approaches that extend high-level representations: Another class of approaches (Hertle *et al.*, 2012; Garrett *et al.*, 2015, 2020) extends the high-level representation to allow the high-level planner to validate preconditions of the high-level actions in the geometric space while computing the high-level plan. Hertle *et al.* (2012) do so by developing *semantic attachments* for *PDDL* representations that check the validity of each high-level action using a motion planner in the low level. *FFRob* (Garrett *et al.*, 2015) uses pre-sampled robot configurations to discretize the problem and build a *roadmap* to evaluate the preconditions of the high-level action. *PDDLStream* (Garrett *et al.*, 2020) uses optimistic samplers to sample continuous arguments in the PDDL descriptions. Their optimistic samplers are analogous to "generators" used by our approach that are used to instantiate abstract actions and serve the same purpose. Our approach and PDDLStream use these samplers to sample concrete values for symbolic abstract arguments.

Approaches that perform an interleaved search: The last group of approaches performs an interleaved search to find a high-level solution that also has valid motion planning refinements at the low level. These approaches incrementally update the high-level models using the feedback from the low-level while searching for the refinements. Srivastava *et al.* (2014) implement a modular approach that uses a planner-independent interface layer to allow communication between a task planner and a motion planner. Dantam *et al.* (2018) develop a constraint-based approach that incrementally adds constraints to the high-level specification of the problem discovered while trying to refine a high-level plan generated

using an *SMT*-based planner. Because these approaches commit to a single high-level model, it is not clear how they would be able to avoid dead ends. Additionally, all these approaches work only for deterministic problems and do not handle stochastic settings.

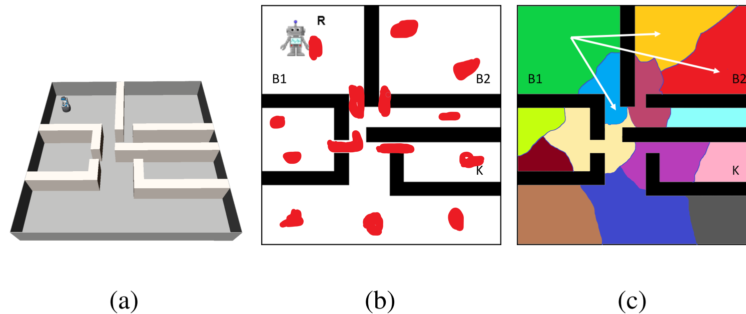
To the best of our knowledge, the only approaches designed to handle stochastic task and motion planning problems were presented by Kaelbling and Lozano-Pérez (2011), Hadfield-Menell *et al.* (2015), and Garrett *et al.* (2020). These approaches consider a partially observable formulation of the problem. Kaelbling and Lozano-Pérez (2011) utilize regression modules on belief fluents to develop a regression-based solution algorithm. Hadfield-Menell *et al.* (2015) extend the work on deterministic task and motion planning by Srivastava *et al.* (2014) for partially observable settings. They use maximum likelihood observations (Platt Jr *et al.*, 2010) to obtain a determinized high-level representation. Garrett *et al.* (2020) develop an online algorithm that uses observational actions to gather beliefs about partially-observable environments and performs task and motion planning using discretized actions. These approaches address a more general class of partially observable problems. However, they do not address the computation of branching policies, which is the key focus of our approach.

3.5 Conclusion

In this chapter, we present a novel anytime approach for task and motion planning when the robot's actions may have multiple outcomes. We provide the nature of abstractions required for our approach and strong theoretical guarantees about the probabilistically-completeness of the algorithms. We show a thorough empirical evaluation of the approach in various settings with simulated and physical robots where approaches that do not explicitly handle stochasticity would have failed to find the solution.

Chapter 4

AUTOMATICALLY LEARNING ZERO-SHOT ABSTRACTIONS FOR DETERMINISTIC MOTION PLANNING



(a) (b) (c)

Figure 4.1: (a) An illustrative environment for a motion planning problem. The robot (R) is tasked to reach the kitchen (K). Red regions (b) show predicted critical regions in the environment. Lastly, (c) shows a projection of the computed state abstraction. Each colored cell represents an abstract state. Arrows show examples of abstract actions, defined as transitions between abstract states.

Chapter 3 shows the effectiveness of abstractions in computing solutions for complex robot planning problems. However, constructing such abstractions requires experts who understand the domain as well as the robot's constraints. This chapter presents our approach for automatically learning such hierarchical state and action abstraction for motion planning problems.

We use the concept of critical regions (Sec. 4.1) for automatically inventing state and action abstractions. Intuitively, critical regions are regions in the environment that are important for solving different robot planning problems. They can be bottleneck states like corridors or hubs such as the center of the room. We then describe our approach for inventing state and action abstractions using critical regions (Sec. 4.2) and later explain our approach – HARP – for using these hierarchical abstractions for robot planning (Sec. 4.3). Sec. 4.4 shows the evaluation of our approach in twenty different environments with four

different holonomic and non-holonomic robots. Lastly, we discuss some approaches that are closely related to our approach (Sec. 4.5).

4.1 Critical Regions

We use the concept of *critical regions* for autonomously inventing state abstractions. Intuitively, critical regions generalize and unify the notions of hubs (e.g., the center of a room from which multiple locations are accessible) and bottlenecks (e.g., a doorway that forces the robot to follow a narrow path). Formally, given a robot r with a configuration space $\mathcal{X}$, Molina *et al.* (2020a) define critical regions as follows.

Definition 6 *Given a robot R, a configuration space $\mathcal{X}$, and a class of motion planning problems M, the measure of criticality of a Lebesgue-measurable open set $r \subseteq \mathcal{X}$ is defined as $\lim_{s_n \to^+ r} \frac{f(r)}{v(s_n)}$, where $f(r)$ is the fraction of observed motion plans solving tasks from M that pass through s_n, $v(s_n)$ is the measure of s_n under a reference density (usually uniform), and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing r ($r \subseteq s_n$, $\forall\, n$).*

Fig. 4.1(b) shows an example of critical regions. We develop an approach that automatically generates a robot-specific, but environment and problem-independent deep neural network architecture and trains it in a self-supervised manner to learn to predict critical regions for unseen environments and problems. We now describe our approach to learning to predict the critical regions.

4.1.1 Learning to Predict Critical Regions

We now present our approach for learning a model Φ that predicts critical regions for a given environment. We first explain how our approach can generate a robot-specific architecture of the network and then describe the training process.

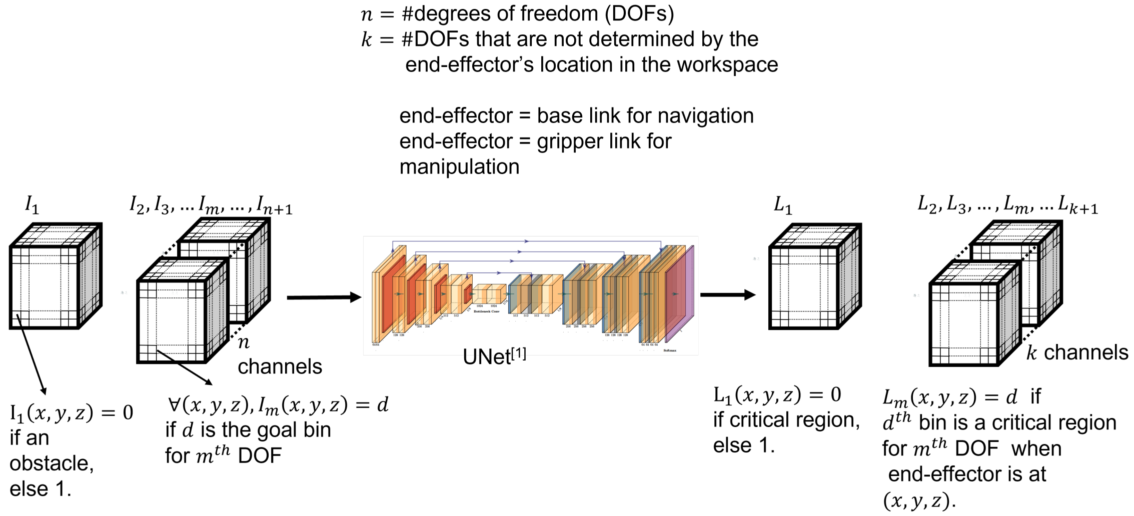


Figure 4.2: Our overall network architecture. Our approach uses the number of degrees of the robot to automatically generate a network architecture that can be used for learning to predict critical regions in unseen test environments.

Deriving robot-specific network architecture

We use the standard fully convolutional UNet architecture (Ronneberger *et al.*, 2015) as our base network. Fig. 4.2 includes network architecture for this network. We use the robot's geometry and its number of DOFs to derive a robot-specific architecture as follows:

Let n be the number of DOFs of the robot and let k be the number of DOFs that are not determined by the location of the robot's end-effector in the workspace. For manipulation problems, we consider the gripper of the robot as its end-effector, and for navigational problems, we consider the robot's base link as its end-effector. First, we use these parameters to update the base UNet architecture. We use the last layer of the base architecture to predict critical regions for the end-effector's location and include k additional convolutional layers to predict critical regions for each of the k DOFs that are not determined by the location of the robot's end-effector.

The input to the network is a tensor of dimension four. The size of the first three dimensions of the input tensor depends on the number of bins used to discretize the environment (which can be arbitrary). The number of channels in the input tensor is determined using

the parameter n. If the robot has n DOFs, then the input tensor would have a total of $n + 1$ channels. The first channel in the input represents the occupancy matrix of the environment. It is generated by performing a raster scan of the environment. The rest of the n channels represent goal values for each DOF of the robot – one for each DOF of the robot.

Similarly, each label is a tensor of dimension four. The size of the first three dimensions is similar to the input tensor. The number of channels in the label tensor is also computed using the robot's geometry. For a robot with k DOFs that are not determined by the location of the robot's end effector in the workspace, the label tensor would have a total of $k + 1$ channels. The first channel represents critical regions for the end-effector's location in the workspace and the rest of the k channels represent critical regions for the k DOFs that are not determined by this location – one channel for each of the k DOFs of the robot.

E.g, consider a 5-DOF hinged robot. The robot's 5 DOFs are $(x, y, z, \theta, \omega)$ where x, y, and z represent the location of the robot's base link in the workspace, θ represents the rotation of the base link, and ω represents the hinged angle. So for this robot, n would equal to 5 and k would equal to 2 as only the base rotation θ and the hinged angle ω are not determined by the location of the robot's end-effector (base link in this case) in the workspace. So according to the previous discussion, the network would contain $k = 2$ additional layers to predict critical regions for θ and ω. The input tensor would have a total of $n + 1 = 6$ channels and the label tensor would have a total of $k + 1 = 3$ channels.

Generating training data

To generate training data for each training environment E, we randomly sample a set of 100 configurations $\mathcal{G}$, which would serve as goal states for the motion planning problems. For each goal configuration $g_i \in \mathcal{G}$, we sample a set of 50 initial states $\mathcal{I}$. We use an off-the-shelf motion planner to compute motion plans for these initial and goal states and combined solutions to generate critical regions for each goal state. We use these motion plans to

compute critical regions for the given pair of environment E and the goal configuration g using the Def. 6. We use OpeanRAVE robot simulator (Diankov, 2010) and OMPL's implementation of BiRRT (Kuffner and LaValle, 2000) to generate the training data.

To generate the input vector, we discretize the environment into n_d bins. This also implies that the degrees of freedom of the robot that are determined by the robot's end-effector's location in the workspace are also discretized into n_d bins. We discretize the rest of the degrees of freedom that are not determined by the robot's end-effector's location into p bins to generate input and label tensors as mentioned earlier. We augment the computed tensors by rotating them by $90°$, $180°$, and $270°$ to obtain more training samples. We also omit an additional dimension from the tensors for navigational problems as we fix the robot's z-axis for these problems. The table below shows the training details for each robot.

We use 20 training environments to generate training data for navigational problems (robots R, Car, and H) and 6 training environments for manipulation problems (Fetch robot). More details on our empirical evaluation are presented in Sec. 4.4. The following table shows input and label tensor shapes as well as the number of training trajectories used for each robot.

Robot	n_d	p	Input Shape	Label Shape	$\|E\|$	# Samples
R, Car	224	4	(224,224,4)	(224,224,2)	20	8000
H	224	5	(224,224,3)	(224,224,21)	20	8000
Fetch	64	10	(64,64,64,11)	(64,64,64,9)	6	720

Training the network

The layer predicting critical regions for the end-effector's locations in the workspace, (L_l), uses the sigmoid activation as the task is similar to element-wise classification. The layers

predicting critical regions for the rest of the degrees of freedom that are not determined by the end-effector's location (L_i) use the softmax activation as the task corresponds to multi-class classification. The loss function is defined as follows:

$$\mathcal{L} = \mathcal{L}_{L_1} + \sum_{i=0}^{k} \mathcal{L}_{L_i}.$$

where $\mathcal{L}_{L_1}$ is weighted log loss and $\mathcal{L}_{L_i}$ is softmax cross entropy loss for i^{th} degree of freedom not determined by end-effecot's location in the workspace. We use *ADAM Optimizer* (Kingma and Ba, 2014) with learning rate 10^{-4}. We implement the UNet architecture shown in Fig. 4.2 using Tensorflow (Abadi *et al.*, 2016) and train it for $50,000$ epochs.

The next section discusses our approach for using automatically predicted critical regions for automatically generating abstract states and actions.

4.2 Inventing State and Action Abstractions

We begin to describe our approach for inventing state and action abstractions with an example. Fig. 4.1(b) shows a set of critical regions for a given environment. Ideally, we would like to predict these critical regions and generate state and action abstractions similar to the one shown in Fig. 4.1(c). The state abstraction shown in Fig. 4.1(c), similar to a Voronoi diagram, generates cells around each critical region such that the distance from each point in a cell to its corresponding critical region is less than that from every other critical region. We call this structure a *region-based Voronoi diagram (RBVD)*. Each cell in this region-based Voronoi diagram is considered an abstract state and transitions between these Voronoi cells (abstract states) define abstract actions.

Let ρ be the set of critical regions for the given configuration space $\mathcal{X}$. First, we introduce the distance metrics d^c and d^r. Here, d^c defines the distance between a low-level configuration $x \in \mathcal{X}$ and a critical region $r \in \rho$ such that $d^c(x,r) = \min_{x_i \in r} d(x, x_i)$ and d^r defines the distance between two critical regions r_1, r_2 such that the distance d^r =

$\min_{x_i \in r_1, x_j \in r_2} d(x_i, x_j)$ where d is the Euclidean distance. Now, we define the region-based Voronoi diagram as follows:

Definition 7 *Let $\rho = \{r_1, ..., r_k\}$ be a set of critical regions for the configuration space $\mathcal{X}$. A region-based Voronoi diagram (RBVD) is a partition $\Psi(\rho, \mathcal{X}) = \{\psi_1, ..., \psi_m\}$ of $\mathcal{X}$ such that for every $\psi_i \in \Psi$ there exists a critical regions r such that for all $x \in \psi_i$ and for all $r_j \neq r$, $d^c(x, r) \leq d^c(x, r_j)$ and each ψ_i is strongly connected.*

State abstraction We define abstract states as the Voronoi cells of an RBVD. Given an RBVD Ψ, the labeling function $\ell : \Psi \rightarrow \mathcal{S}$ maps each cell in the RBVD to a unique abstract state $s \in \mathcal{S}$ where $|\mathcal{S}| = |\Psi|$. We use this to define the state abstraction function α as follows:

Definition 8 *Let R be the robot and $\mathcal{X} = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ be the configuration space of the robot R with a set of critical regions ρ. Let $\Psi(\rho, \mathcal{X}) = \{\psi_1, ..\psi_k\}$ be an RBVD for the robot R, configuration space $\mathcal{X}$, and the set of critical regions ρ and let $gS = \{s_1, .., s_k\}$ be a set of high-level, abstract states. We define abstraction function $\alpha : \mathcal{X}_{\text{free}} \rightarrow \mathcal{S}$ such that $\alpha(x) = s$ where $x \in \psi$ and $\ell(\psi) = s$.*

We extend this notation to define membership in abstract states as follows: Given a configuration space $\mathcal{X} = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ and its set of abstract states $\mathcal{S}$ as defined above, a configuration $x \in \mathcal{X}_{\text{free}}$ is said to be a member of an abstract state $s \in \mathcal{S}$ (denoted $x \in s$) iff $\alpha(x) = s$. We also extend the notion of strong connectivity to abstract states as follows: An abstract state $s \in \mathcal{S}$ is strongly connected iff $\ell^{-1}(s)$ is strongly connected. We now define adjacency for Voronoi cells in a region-based Voronoi diagram as follows. Recall that C denotes Euclidean connectivity for configurations.

Definition 9 *Let ψ_i, ψ_j be Voronoi cells of an RBVD Ψ. Voronoi cells ψ_i and ψ_j are adjacent iff there exist configurations x_i, x_j such that $x_i \in \psi_i$, $x_j \in \psi_j$, $C(x_i, x_j) = 1$, and there exists a trajectory π between x_i and x_j such that $\forall t \in [0, 1], \pi(t) \in \psi_i$ or $\pi(t) \in \psi_j$.*

We extend the above definition to define the neighborhood for an abstract state. Two abstract states $s_i, s_j \in \mathcal{S}$ are neighbors iff $\ell^{-1}(s_i)$ and $\ell^{-1}(s_j)$ are adjacent.

We define abstract actions as transitions between abstract states. Let $\mathcal{S}$ be the set of abstract states. We define the set of abstract actions $\mathcal{A}$ using $\mathcal{S}$ such that $\bar{\mathcal{A}} = \{a_{ij} | \forall \ (s_i, s_j) \in \mathcal{S} \times \mathcal{S}\}$.

We now use this formulation of RBVD and state abstraction to prove the soundness of the generated abstractions.

Theorem 4.2.1 *Let $\mathcal{X} = \mathcal{X}_{free} \cup \mathcal{X}_{obs}$ be a configuration space and ρ be a set of critical regions for $\mathcal{X}$. Let Ψ be an RBVD for the critical regions ρ and the configuration space $\mathcal{X}$ and let $\mathcal{S}$ be the set of abstract states corresponding to Ψ with a mapping function ℓ. Let x_0 and x_g be the initial and goal configurations of a holonomic robot R. If every state $s \in \mathcal{S}$ is strongly connected and there exists a sequence of abstract states $P = \langle s_{\psi_0}, ..., s_{\psi_g} \rangle$ such that $x_0 \in s_{\psi_0}$, $x_g \in s_{\psi_g}$, and all consecutive states $s_{\psi_i}, s_{\psi_{i+1}} \in P$ are neighbors, then there exists a motion plan for R that reaches x_g from x_0 with a trajectory π such that $\pi(0) = x_0$, $\pi(1) = x_g$, and $\forall x_i \in \pi, x_i \in s_{\psi_k}$ such that $s_{\psi_k} \in P$.*

Proof 4.2.1 *For two consecutive abstract states $s_i, s_{i+1} \in P$, let $\psi_i, \psi_{i+1} \in \Psi$ be Voronoi cells such that $\ell^{-1}(s_i) = \psi_i$ and $\ell^{-1}(s_{i+1}) = \psi_{i+1}$. If s_i and s_{i+1} are neighbors, then according to Def. 9 there exists a pair of low-level configurations $x_i, x_{i+1} \in \mathcal{X}_{free}$ such that there exists a collision free trajectory between x_i to x_{i+1}. Def. 7 defines every Voronoi cell as a strongly connected set. Thus, for every low-level configuration $x_j \in s_i$, there exists a collision-free trajectory between x_j and x_i and for every low-level configuration $x_k \in s_{i+1}$, there exists a collision-free trajectory between x_k and x_{i+1}. For a holonomic robot R these trajectories should be realizable as all degrees of freedom of R can be controlled independently. This implies that there exists a motion plan for R between each pair of configurations in s_{ψ_i} and $s_{\psi_{i+1}}$.*

Algorithm 2: Hierarchical Abstraction-guided Robot Planner (HARP)

Input: Configuration space $\mathcal{X}$, a region predictor Φ, an initial configuration $x_0 \in \mathcal{X}$, goal configuration $x_g \in \mathcal{X}$, a custom heuristic h, low-level sampling-based motion planner *MP*

Output: A motion plan π

1 $\rho \leftarrow$ predict_critical_regions(Φ, $\mathcal{X}$, x_0, x_g)
2 $\mathcal{S}, \mathcal{A} \leftarrow$ generate_state_action_abstractions(ρ, $\mathcal{X}$)
3 $s_0, s_g \leftarrow$ get_HL_state($\mathcal{S}$, ρ, x_0), get_HL_state($\mathcal{S}$, ρ, x_g)
4 $\mathcal{P} \leftarrow$ multi-source_bi-directional_beam_search($\mathcal{S}$, $\mathcal{A}$, s_0, s_g)
5 $\pi \leftarrow$ refine_path($\mathcal{P}$, *MP*)
6 $h \leftarrow$ update_heuristic(π, $\mathcal{S}$)

7 return π

Theorem 4.2.1 proves that the computed abstractions would be sound as well as satisfy the downward refinement property for holonomic robots. The proof does not hold for non-holonomic robots as the low-level trajectories may not be realizable given their motion constraints. However, the algorithm developed below is probabilistically complete for all robots and performed well for non-holonomic robots in our empirical evaluation.

4.3 Hierarchical Abstraction-Guided Robot Planner (HARP)

In this section, we describe our approach – ***H**ierarchical **A**bstraction-guided **R**obot **P**lanner (HARP)* – for generating abstract states and actions and using them to efficiently perform hierarchical planning. A naíve approach would be to generate a complete RBVD and then extract abstract states and actions from it. This would require iterating over all configurations in the configuration space and computing a large number of motion plans to identify executable abstract actions. This is expensive (and practically infeasible) for continuous low-level configuration spaces. Instead, we use the RBVD as an implicit concept. We generate abstractions on-the-fly by computing membership of low-level configurations in abstract states only when needed.

Vanilla high-level planning using the set of all abstract actions $\mathcal{A}$ would be inefficient

as it may yield plans for which low-level refinement may not exist as we do not know the applicability of these abstract actions at the low level. To overcome this challenge, we develop a hierarchical multi-source bi-directional planning algorithm that performs high-level planning from multiple abstract states. Generally, a multi-source approach would not work for robot planning because it is not clear what the intermediate states are. HARP, on the other hand, uses critical regions as abstract intermediate states and utilizes a multi-source search. This utilizes learned information better than single source and single direction beam search. Our high-level planner generates a set of candidate high-level plans from the abstract initial state to the abstract goal state using a custom heuristic (which is continually updated). These paths are then simultaneously refined by a low-level planner to compute a trajectory from the initial low-level configuration to the goal configuration while updating the heuristic function.

Algorithm 2 describes our approach for generating and using hierarchical abstractions. Given the configuration space $\mathcal{X}$ and initial and goal configurations (x_0 and x_g) of the robot R, HARP uses a learned DNN Φ to generate a set of critical regions ρ (line 1). The remainder of Alg. 2 can be broken down into three important steps: 1) computing a set of candidate high-level plans, 2) refining candidate high-level plans into a low-level trajectory, and 3) updating the heuristic for abstract states. We now explain each of these steps in detail.

Computing high-level plans To compute high-level plans that reach the goal configuration x_g from the initial configuration x_0, first we determine abstract initial and goal states s_0 and s_g corresponding to the initial and goal configurations x_0 and x_g (line 3). To do this efficiently, we store sampled points from each critical region in a K-D tree and query it to determine the abstract state for the given low-level configurations without explicitly constructing the complete RBVD. This allows us to dynamically and efficiently determine

Algorithm 3: Multi-source Bi-directional beam search

Input: Set of states $\mathcal{S}$, set of actions $\mathcal{A}$, initial state s_0, goal state s_g, distance function h', beam width w, number of high-level plans N

Output: A set of N paths from s_0 to s_g

```
1  fringe = PriorityQueue()
2  fringe.add(s_0), fringe.add(s_g)
3  S̄ ← sample_states(S)
4  solutions = Set()
5  foreach s ∈ S̄ do
6      fringe.add(0,(s, None, Set()))
7  while N > 0 and fringe is not empty do
8      working_fringe ← select top w nodes from the fringe
9      empty_fringe(fringe)
10     while working_fringe is not empty do
11         current, path, visited ← working_fringe.pop()
12         if current = s_g then
13             add path to solutions
14             N ← N − 1
15         else
16             path.add(current)
17             visited.add(current)
18             foreach node ∈ current.successors do
19                 if node ∉ visited then
20                     p ← h'(current, node) + min{h'(node, s_0), h'(node, s_g)}
21                     fringe.add(p, (current, path, visited))
```

high-level states for low-level configurations.

Once we determine initial and goal states s_0 and s_g, we use our high-level planner to compute a set of candidate high-level plans going from s_0 to s_g (line 4). To compute these candidate high-level plans, we develop a multi-source bi-directional variant of beam search (Lowerre, 1976) that yields multiple high-level candidate plans. We call this *multi-source bi-directional beam search*. Alg. 3 presents the pseudocode for multi-source bi-directional beam search.

Intuitively, this algorithm works as follows: we use a priority queue to maintain a fringe to keep track of the current state of the search. We initialize this fringe with multiple

randomly sampled abstract states (line 5) to allow the beam search to start from multiple sources. We select and expand these nodes in a specific order (line 11) to compute high-level plans that reach from the initial state to the goal state.

Formally, the nodes in the fringe are expanded as follows: Let n be a node in the fringe. We select the node to expand using $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path heading to n (unit cost for each action) and $h(n)$ is a custom heuristic that is defined as follows. Let m be the parent of n and let s_n and s_m be the abstract states corresponding to the nodes n and m. The heuristic $h(n)$ is computed as:

$$h(n) = h'(s_m, s_n) + min\{h'(s_n, s_i), h'(s_n, s_g)\}.$$

Here, $h'(s_1, s_2) = \epsilon_{12} d^r(r_1, r_2)$ defines the estimated distance between abstract high-level states s_1 and s_2 with corresponding critical regions r_1 and r_2 respectively. s_i is the initial abstract state and s_g is the goal abstract state.

$\epsilon_{ij} \in (0, 1]$ is a constant that accounts for imprecise abstract actions. Alg. 2 dynamically changes it to update the heuristic function (HARP line 6) and making it more accurate. Initially, ϵ_{ij} is set to 1 for all i and j. Once a low-level trajectory π is computed (explained later), we compute the abstraction $\bar{\pi}$ of this trajectory. For each consecutive pair of abstract states $\langle s_i, s_j \rangle$ in $\bar{\pi}$, we decrease the value of ϵ_{ij} by $\epsilon_{ij}/2$. This allows HARP to use experience from previously computed trajectories to prioritize abstract actions that have low-level refinements to compute accurate high-level plans.

After an abstract state s_j is selected from the fringe (Alg. 3, line 8), its successors are created by applying abstract actions on it and adding these successors to the fringe (Alg. 3 lines 18-21). This process continues until N high-level plans are found.

Once we generate a set of candidate high-level plans from multi-source bi-directional beam search, we use a low-level planner to refine these plans into a low-level collision-free trajectory from initial low-level configuration x_0 to goal configuration x_g (line 5).

Refining high-level plans While any *probabilistically complete* motion planner can be used to refine the computed high-level plans into a low-level trajectory between given two configurations, we use *Learn and Link Planner (LLP)* (Molina *et al.*, 2020b) as a low-level planner in HARP (*MP* in Alg. 2) as it allows us to easily use the abstract states and their critical regions to efficiently compute motion plans. LLP is a sampling-based motion planner that initializes exploration trees rooted at N samples from the configuration space and extends these exploration trees until they connect and form a single tree. Once a single tree is formed, the planner uses Dijkstra's Algorithm (Dijkstra, 1959) to compute a path from the initial state to the goal state.

To use LLP to refine a set of candidate high-level plans simultaneously, we first select a subset of critical regions $\bar{\rho} \subseteq \rho$ that includes all critical regions corresponding to all high-level states in candidate plans. We use this subset of critical regions $\bar{\rho}$ to provide initial samples to initialize exploration trees of LLP. In this work, we generate M samples from the set of critical regions $\bar{\rho}$ and generate the rest of the $N - M$ samples using uniform random sampling. Similarly, to expand these exploration trees, we generate a fixed number of samples from the set critical regions ρ and then continue with uniform sampling.

We use these characteristics of our algorithm to show that our approach is *probabilistic complete*.

Theorem 4.3.1 *If the low-level motion planner (*MP *in Alg. 2) is probabilistically complete, then HARP is probabilistically complete.*

Proof 4.3.1 *(Sketch) While refining high-level plans to a low-level motion plan (line* 5 *in Alg. 2), HARP uses a fixed number of samples from the critical regions along the high-level plans to initialize a low-level motion planner. This does not reduce the set of support (regions with a non-zero probability of being sampled) of the sampling distribution being used by the motion planner.*

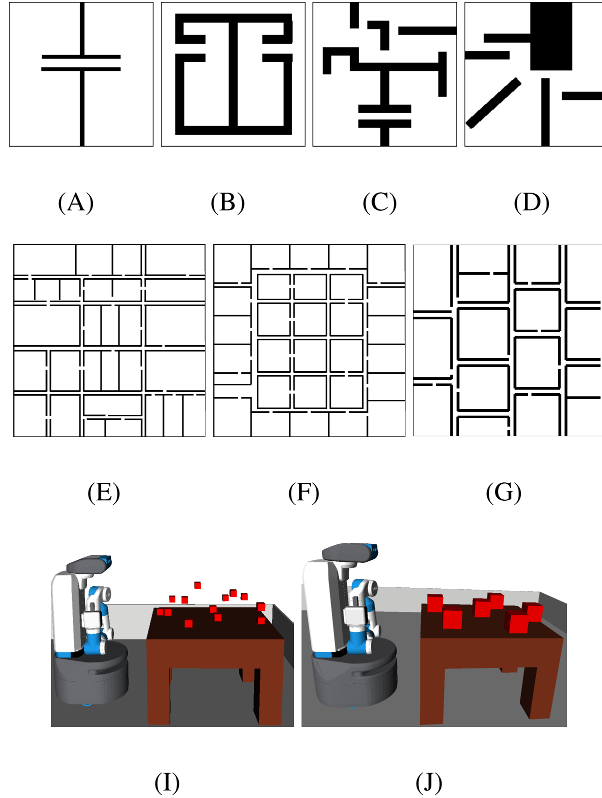


Figure 4.3: Test environments for our approach. Dimensions of environments (A)-(D) are $\mathbf{5m} \times \mathbf{5m}$ and dimensions of environments (E)-(G) are $\mathbf{25m} \times \mathbf{25m}$. (A)-(G) are used for navigational problems while (H) and (I) are used for manipulation problems.

4.4 Empirical Evaluation

We extensively evaluate our approach in twenty different scenarios with four different robots. All experiments were conducted on a system running *Ubuntu 18.04* with *8-core i9* processor, *32 GB RAM*, and an *Nvidia 2060* GPU (our approach uses only a single core) OpenRAVE robot simulator (Diankov, 2010). We compare our approach with state-of-the-art motion planners such as *RRT* (LaValle, 1998), *PRM* (Kavraki *et al.*, 1996), and *BiRRT* (Kuffner and LaValle, 2000). As LLP is implemented using *Python*, we use the *Python* implementation of the baseline algorithms available at `https://ompl.kavrakilab.org/` for comparison. Our training data, python implementation, trained models, and results are available at `https://aair-lab.github.io/harp.html`.

3-DOF rectangular robot (R) For the first set of experiments, the objective is to solve motion planning problems for a 3-DOF rectangular robot. The robot can move along the x and y axes, and it can rotate around the z axis.

3-DOF non-holonomic rectangular car robot (Car) For the second set of experiments, we evaluated our approach with a rectangular non-holonomic robot similar to a simple car. Controls available to operate the robot were linear velocity $v \in [-0.2, 0.2]$ and the steering angle $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ while three degrees of freedom (location along x-axis, location along y-xis, and rotation around z-axis) were required to represent the robot's transformation.

4-DOF hinged robot (H) For the third set of experiments, we used a robot with a hinge joint to evaluate our approach. The robot's 4 DOFs are its location along x and y axes, rotation along z-axis (θ), and the hinge joint (ω) with the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

8-DOF fetch robot For the last set of experiments, we used our approach with a mobile manipulator named Fetch (Wise *et al.*, 2016) to perform arm manipulation. The goal of this experiment is to evaluate the scalability of our approach to robots with high degrees of freedom.

Evaluating the approach Figures 4.3 show the test environments (unseen by the model while training) for our system. Environments shown in Fig. 4.3 are inspired by the indoor office and household environments. Our training data consisted of 20 environments similar to the ones shown in Fig. 4.3(A)-(D) with dimensions $5m \times 5m$. We investigate the scalability of our approach by conducting experiments in environments shown in Fig. 4.3(E)-(G) with dimensions $25m \times 25m$ (much larger than training environments). To handle such large environments without making any changes to the DNN, we use the standard approach of sliding windows with stride equal to window-width (Tang *et al.*, 2020; Birgui Sekou *et al.*,

2018; Lu *et al.*, 2015; Hou *et al.*, 2016). This crops the larger environment into pieces of the size of the training environments. Individual predictions are then combined to generate a set of critical regions for arbitrarily large environments. We also evaluate the applicability of our approach to non-holonomic robots in environments shown in Fig. 4.3(A)-(D).

4.4.1 Analysis of the Results

The main objective of our empirical evaluation is to show whether $1)$ state and action abstractions can be derived automatically and $2)$ whether auto-generated state and action abstractions can be efficiently used in a hierarchical planning algorithm. Additionally, we also investigate $3)$ does dynamically updating the heuristic function (line 6 in Alg. 2) improve Alg. 2's efficiency?

State and action abstractions Our approach learns critical regions for each DOF of the robot. Fig. 4.4 and Fig. 4.5 show critical regions predicted by our learned model for the hinged robot H. We can see that our model was able to identify critical regions in the environment such as doorways and narrow hallways. Fig. 4.4(a) and 4.5(a) show the test environments; Fig. 4.4(b) and Fig. 4.5(b) show the predicted critical regions for the x and y location of the base of the robot and Fig. 4.4(c) and Fig. 4.5(c) critical regions for orientation of the robot's base link (captured by DOF θ). The blue regions in the figure represent the horizontal orientation of the robot and the green regions represent the vertical orientation of the robot. Fig. 4.5(d) shows critical regions for the hinge joint ω for the robot H. Here, blue regions show that the network predicted the hinge joint to be flat (close to 0°) and green regions represent configurations where the model predicted "L" configurations of the robot (ω close to 90° or 270°). Fig. 4.4 shows that our approach was able to predict the correct orientation of the robot accurately most of the time.

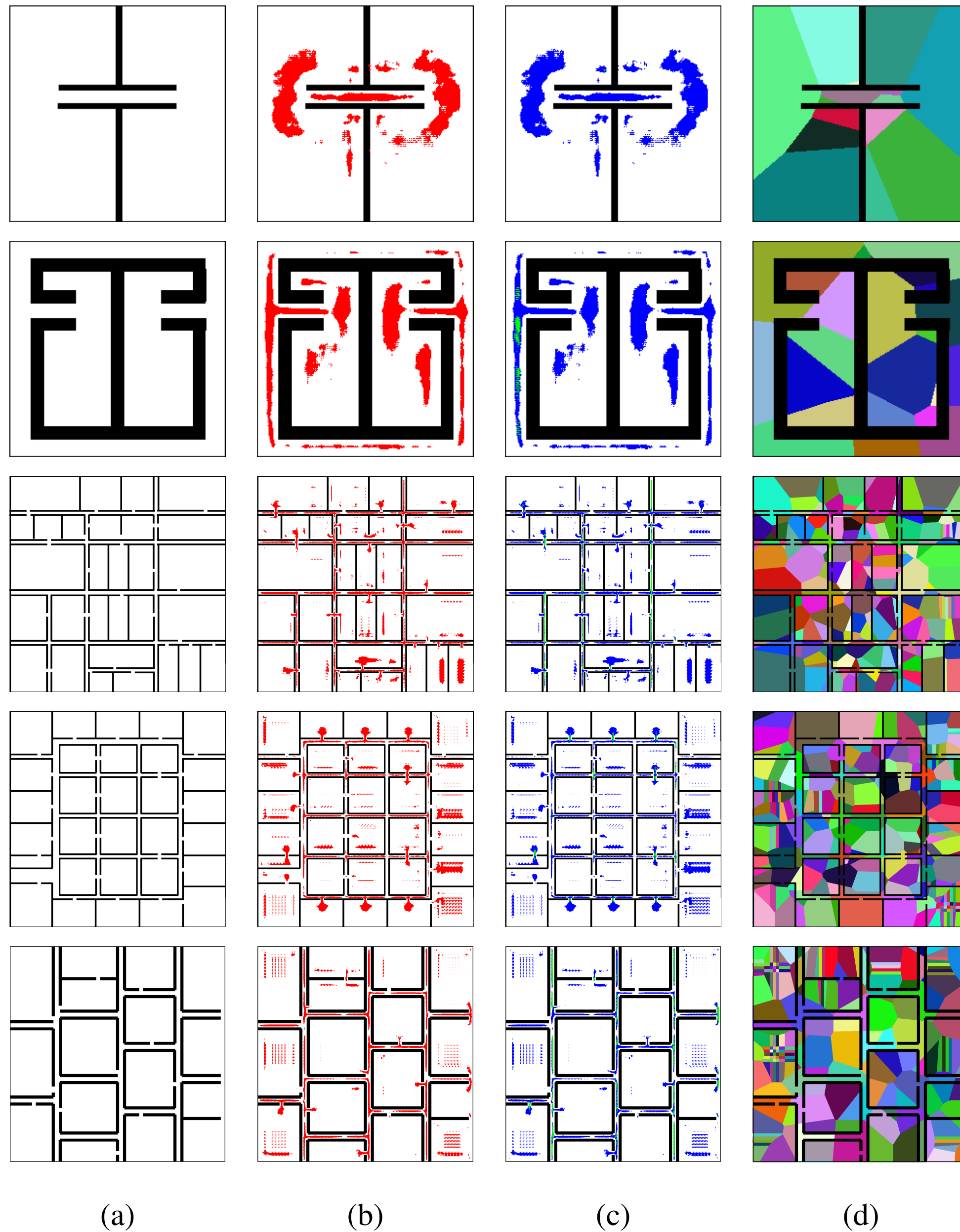


Figure 4.4: Predicted critical regions and computed state abstractions for 3-DOF rectangular (R) and Car robots. (a) Input to the environment. (b) Critical regions for the location of the robot's base link in the workspace. (c) Critical regions for the rotation of the robot's base link of the robot. Blue regions are locations where the network predicted the robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical. (d) 2D projections of state abstraction generated by our approach. State abstractions are strictly for visualization as our approach does not require generating them.

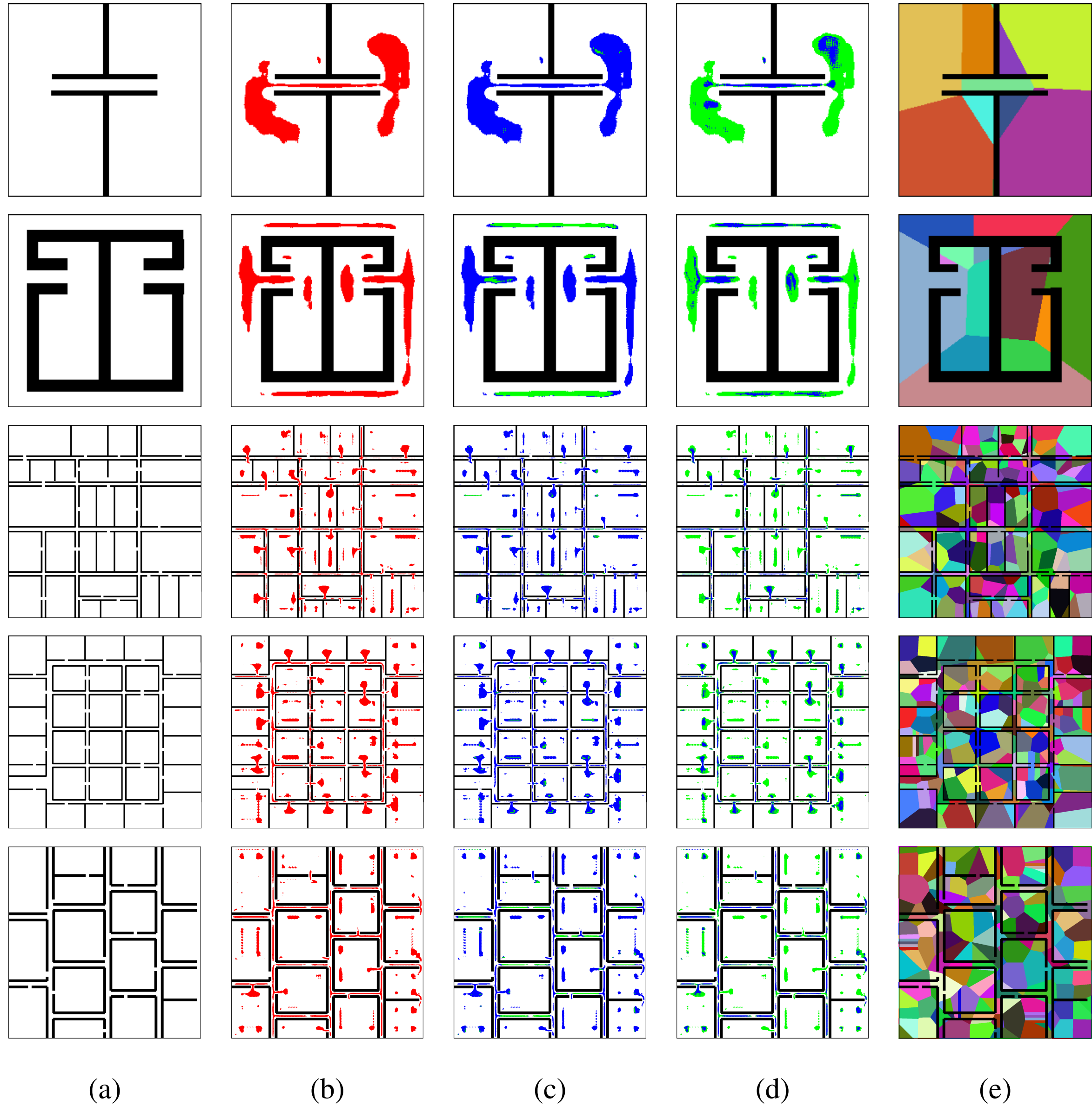


Figure 4.5: Predicted critical regions and generated state abstractions for a 4-DOF hinged robot. (a) Input to the environment. (b) Critical regions for the location of the robot's base link in the workspace. (c) Critical regions for the orientation of the robot's base link. Blue regions are locations where the network predicted the robot to be horizontal and green regions are the regions where the network predicted the robot to be vertical. (d) Critical regions for the hinge joint. Blue regions show that the network predicted it to be closer to $180°$ and green regions show that the network predicted the hinge angle close to $90°$ or $270°$. (e) 2D projections of state abstraction generated by our approach. State abstractions are strictly for visualization as our approach does not require to generate this explicitly.

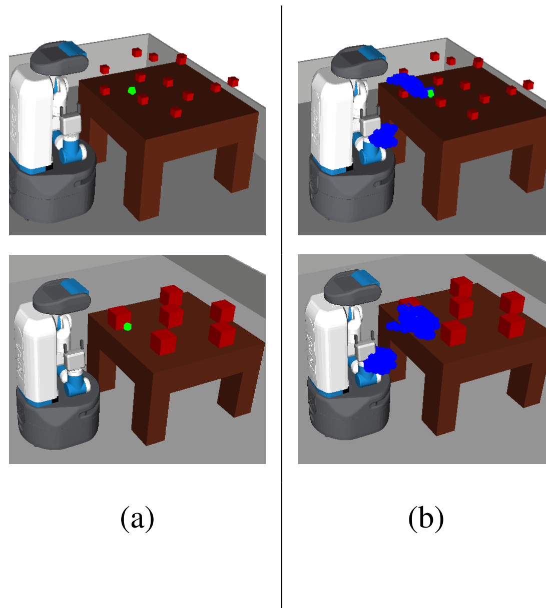


Figure 4.6: Predicted critical regions for an 8-DOF Fetch robot. The green region in (a) shows the goal location for the end effector. (b) shows the critical regions generated by the learned model. Although the network predicts critical regions for all the joints, only critical regions or end-effector's location in the workspace are shown.

Our approach was able to scale to robots with a high number of degrees of freedom. Fig. 4.6(a) shows one of the test environments used for these experiments and Fig. 4.6(b) shows the predicted critical regions. This shows that our model was able to learn critical regions in the environment that can be used to generate efficient abstractions.

Now we answer the second question on whether using abstractions to compute motion plans helps improve the planner's efficiency by qualitatively comparing our approach with a few existing sampling-based motion planners.

Efficiency of planning with learned abstractions We compare our approach against widely used SBMPs such as RRT (LaValle, 1998), PRM (Kavraki *et al.*, 1996), and BiRRT (Kuffner and LaValle, 2000).

Fig. 4.7 shows the comparison of our approach with other sampling-based motion planners. The x-axis shows the time limit in seconds and the y-axis shows the percentage of

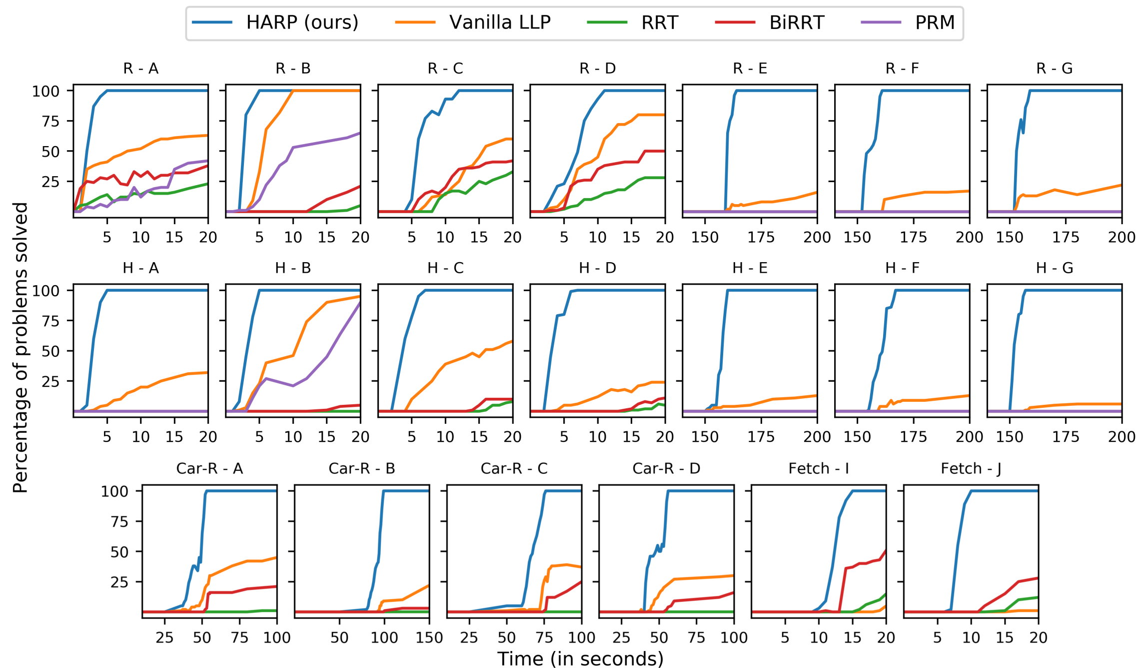
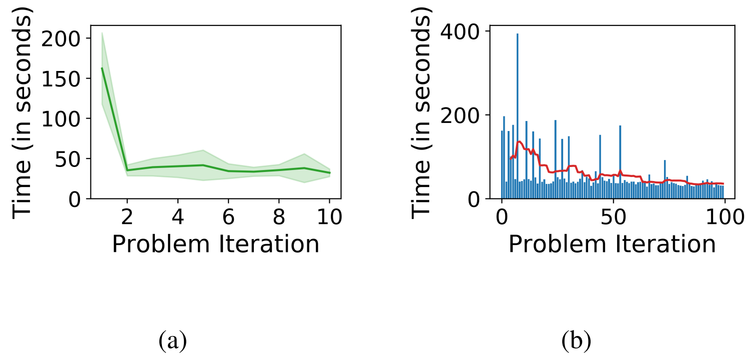


Figure 4.7: Each plot shows the fraction of **100** independently generated motion planning tasks solved (y-axis) in the given time (x-axis) for all the test environments and robots. The title of each subplot represents the robot and the environment. E.g., "R - A" stands for rectangular robot in environment A (Fig. 4.3(A)) and "H - A" stands for hinged robot in environment A.

motion planning problems solved in that time limit. For each time limit on the x-axis, we randomly generate 100 new motion planning problems to thoroughly test our approach and reduce statistical inconsistencies. Fig. 4.7 shows that our approach significantly outperforms all of the existing sampling-based motion planners. Specifically for environments C, D, and E, uniform sampling-based approaches were not able to solve a single problem in a time threshold of $600s$.

Our approach also outperforms the learning-based planner LLP (Molina *et al.*, 2020b) (Fig. 4.7), which uses learned critical regions but does not use state and action abstractions and does not perform hierarchical planning. This illustrates the value of learning abstractions and using them efficiently for hierarchical planning.

Similarly, we also evaluate our approach against TogglePRM (Denny and Amato, 2013).



(a) (b)

Figure 4.8: (a) Solving **20** randomly generated problems repeatedly 1**10** times. The x-axis shows the problem iteration and the y-axis shows the average time over **20** problem instances. (b) Time taken to solve **100** randomly generated problem instances. The X-axis shows the problem number and the y-axis shows the taken to solve each problem.

TogglePRM is written in C++ and accepts only discrete SE^2 configuration space for a simple dot robot. We created discrete variants of environments shown in Fig. 4.3(A) and 4.3(B) with a total of 50176 states and compared the total number of nodes sampled. For 100 random trials, on an average, our approach generated 631 ± 278 and 496 ± 175 states compared to TogglePRM which generated 4234 ± 532 and 19234 ± 4345 states for discrete variants of the environments shown in Fig. 4.3(A) and 4.3(B) respectively. Our approach was able to outperform TogglePRM since these environments do not have α-ϵ-separable passages (Denny and Amato, 2013).

Effectiveness of dynamically updating the heuristic We carried out two sets of experiments. In the first set of experiments, we generated 20 random motion planning problems and solved each problem repeatedly for 10 times while updating the heuristic function. We maintained separate copies of high-level heuristic functions for each problem. Fig. 4.8(a) shows the results for this set of experiments in the environment E (Fig. 4.3(E)) with the 4-DOF hinged robot. The x-axis shows the planning iteration and the y-axis shows the average time over randomly generated 20 problem instances. We can see how planning time reduces drastically once costs for abstract actions are updated.

In the second set of experiments, we generated 100 random pairs of initial and goal states and computed motion plans for each of them. This time, we maintained a single heuristic function across all problems and updated it after each motion planning query. Fig. 4.8(b) shows the result of the experiment in the environment E (Fig. 4.3(E)) with the 4-DOF hinged robot H. The x-axis shows the problem number and the y-axis shows the time taken by our approach to compute a solution. The red line in the plot shows the moving average of planning time. Fig. 4.8(b) shwows that dynamically updating the heuristic function for high-level planning helps to increase the efficiency of HARP and decrease motion planning times.

Empirical evaluation using these experiments validates our hypothesis that learning abstractions and effectively using them improves motion planning efficiency.

Now, we discuss closely related approaches.

4.5 Related Work

Much of the prior work on the topic is focused on decomposing a motion planning problem into smaller subproblems to reduce its complexity. Several approaches have been proposed that use state decomposition to reduce the complexity of a motion planning problem. *Vertical cell decomposition* (Chazelle, 1985) partitions the state space into a collection of vertical cells and computes a roadmap that passes through all of these cells. Brock and Kavraki (2001) propose a hierarchical method that uses *wavefront expansion* to compute the decomposition of the state space. While these approaches establish the foundation of decomposition-based motion planning, partitions generated through such approaches are arbitrary and do not provide any guarantees of completeness. Şimşek *et al.* (2005) use graph cut over local transition graph to identify interface points between highly dense regions. They use these interface points to learn options that take the agent from one region to another region. One of the major distinctions between theirs and our approach is that

our approach operates with continuous state and action spaces, but their approach requires discrete actions. Their approach also requires collecting local experience for every new environment while our approach learns the model that identifies critical regions once and uses the same model for every new environment. Additionally, their approach is *dual* to our approach as it aims to identify interface points between regions while our approach aims to directly identify these regions in the environment using a pre-trained network and does not need to identify interface points.

Zhang *et al.* (2018) use rejection sampling to reject unrelated samples to speed up SBMPs.They use reinforcement learning to learn a policy that decides to accept or reject a new sample to expand the search tree. While their approach reduces the search space to compute the path, it still needs to process samples generated from regions that are irrelevant to the current problem. On the other hand, our hierarchical approach refines abstract plans into low-level motion plans which reduces the number of unnecessary samples. TogglePRM (Denny and Amato, 2013) maintains roadmaps for free space and obstacle space in the configuration space to estimate the narrow passages and sample points from these narrow passages. This approach works well for environments with α-ϵ-separable passages, even though it does not compute high-level abstractions.

Multiple approaches have used statistical learning to boost motion planning. Wang *et al.* (2021) present a comprehensive survey of methods that utilize a variety of learning methods to improve the efficiency of SBMPs. Multiple approaches discussed by Wang *et al.* (2021) use end-to-end deep learning to learn low-level reactive policies. End-to-end approaches are attractive given if they succeed, they can compute solutions much faster than traditional approaches, but it is not exactly clear under which conditions these algorithms would succeed. Formally, these end-to-end deep learning-based approaches lack the guarantees of completeness and soundness that our approach provides. Wang *et al.* (2021) also discuss approaches that use learning to aid sampling-based motion planning. We discuss

a few of these approaches that are relevant to this work. Kurutach *et al.* (2018) uses *Info-GAN* (Chen *et al.*, 2016) to learn state-space partitioning for simple SE^2 robots. While their empirical evaluation shows promising results, similar to previous decomposition-based approaches, they do not provide any proof of completeness. It is also not clear how their approach would scale to configuration spaces that had more than two dimensions. On the other hand, our approach provides formal guarantees of completeness and soundness (for holonomic robots) and scales to high-dimensional spaces.

Ichter *et al.* (2018) and Kumar *et al.* (2019) use a conditional variational autoencoder (CVAE) (Sohn *et al.*, 2015) to learn sampling distributions for the motion planning problems. Ichter *et al.* (2020) use *betweenness centrality* (Şimşek *et al.*, 2005) to learn criticality score for low-level configurations. They uniformly sample a set of configurations from the environment and use configurations with higher criticality from this set to generate a roadmap. Their results show significant improvement over vanilla PRM but it is unclear how their approach would perform if the environment had regions that are important to compute motion plans yet difficult to sample under uniform sampling. On the other hand, our approach would identify such important regions to overcome these challenges. While these approaches (Ichter *et al.*, 2018; Kumar *et al.*, 2019; Ichter *et al.*, 2020) focus on biasing the sampling distribution towards narrow areas in the environment, our approach aims to build more general high-level abstractions for the configuration space.

Molina *et al.* (2020b) use an image-based approach to learn and infer the sampling distribution using demonstrations. They use top-view images of the environment with *critical regions* highlighted in the image to learn to identify critical regions. While they develop a method for predicting critical regions and using them with a low-level motion planner, they do not use these critical regions for learning abstractions and performing hierarchical planning. Our empirical results (Sec. 4.4) show that our hierarchical approach is much more effective than their non-hierarchical approach and yields significantly better performance.

Additionally, their approach is also restricted to navigational problems and does not scale to configuration spaces with more than two degrees of freedom (DOFs).

Deep learning has also been used for learning heuristics for high-level symbolic planning. Shen *et al.* (2020) use hypergraph networks for learning heuristics for symbolic planning in the form of delete-relaxation representation of the actual planning problems. Karia and Srivastava (2021) learn generalizable heuristics for high-level planning without explicit action representations in symbolic logic. In contrast, we focus on learning critical regions and creating high-level abstractions along with algorithms that work with these learned high-level abstractions.

Liu *et al.* (2020) use semantic information to bias the sampling distribution for navigational problems in partially known environments. Compared to it, our approach is not navigational problems and does not require semantic information explicitly but aims to learn such a notion in the form of critical regions. *SPARK* and *FLAME* (Chamzas *et al.*, 2020) use state decomposition to store past experience and use it when queried for similar state decompositions. While their approach efficiently uses the experience from previous iterations, it requires carefully crafted state decompositions to cover a large number of scenarios, whereas our approach generates state abstraction automatically using the predicted critical regions.

4.6 Conclusion

In this chapter, we presented a probabilistically complete approach, HARP, that uses deep learning to identify abstractions for the input configuration space. It invents state and action abstractions in a bottom-up fashion and uses them to perform efficient hierarchical robot planning. We develop a novel multi-source bi-directional planning algorithm that uses learned state and action abstractions along with a custom dynamically maintained cost function to generate candidate high-level plans. A low-level motion planner refines

these high-level plans into a trajectory that achieves the goal configuration from the initial configuration.

Our approach provides a way to generate sound abstractions that satisfy the downward refinement property for holonomic robots. Our empirical evaluation on a large variety of problem settings shows that our approach can significantly outperform state-of-the-art sampling and learning-based motion planners. Through our empirical evaluation, we show that our approach is robust and can be scaled to large environments and to robots that have high degrees of freedom. Our work presents a foundation for learning high-level, abstractions from low-level trajectories. Currently, our approach works for deterministic robot planning problems.

Chapter 5

ZERO-SHOT OPTION INVENTION FOR STOCHASTIC MOTION PLANNING PROBLEMS

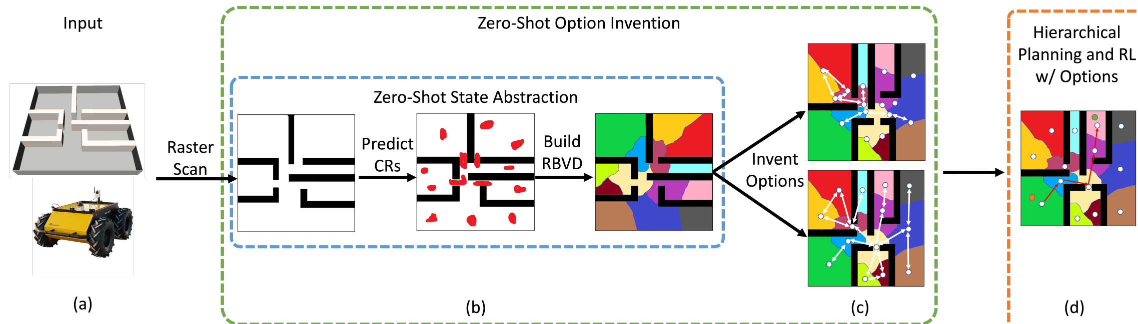


Figure 5.1: Our overall approach for automatically inventing high-level options. (a) shows the input to our system. (b) shows the zero-shot abstraction process along with the raster scan of the input environment (left), critical regions predicted by the learned network (center), and the zero-shot state abstraction (right). The top image in (c) shows a subset of automatically invented interface options and the bottom image shows a subset of automatically invented centroid options. Lastly, these learned options are used for hierarchical planning and learning. The red arrows in (d) show an example of a high-level plan over centroid options given an initial configuration (orange area) and a goal configuration (green area). Policies for these options are learned using deep reinforcement learning and the auto-generated dense pseudo-reward function.

Motion planning assumes perfect controllers that can control a robot ideally without any stochasticity. However, in the real world, controllers are often stochastic and noisy. Therefore, it is infeasible to accurately follow a single motion plan and instead, the robot requires a motion policy that takes the robot to the goal configuration while handling the stochasticity. In this chapter, we present our approach to automatically inventing abstractions for computing such motion policies.

Fig. 5.1 shows our overall approach. To generate state abstractions, our approach uses methods developed in Ch. 4 off-the-shelf and uses these state abstractions for autonomously inventing high-level options. The next sections in the chapter outline our approach for

Algorithm 4: OptionInventor

Input: robot R, training environments E_{train}, test environment E_{test}
Output: set of option $\mathcal{O}$, cost function C

1. $\Theta \leftarrow$ get_critical_region_predicter(R);
2. **if** Θ *is not trained* **then**
3. train Θ using E_{train}
4. $\Phi \leftarrow$ predict_critical_regions(E_{test},Θ);
5. Ψ, $\mathcal{S}$, $\mathcal{V} \leftarrow$ construct_RBVD(E_{test},Φ);
6. $\mathcal{O}$, $C \leftarrow$ construct_options(Ψ,$\mathcal{S}$,$\mathcal{V}$);
7. **foreach** $o \in \mathcal{O}$ **do**
8. mp$_o \leftarrow$ compute_motion_plan(o);
9. $\mathcal{G}_o \leftarrow$ compute_option_guide(o,mp$_o$);
10. return $\mathcal{O}$, C

automatically inventing high-level options (Sec. 5.1) and a novel approach for combining symbolic planning with hierarchical RL (Sec. 5.2) along with strong theoretical guarantees. Sec. 5.3 provides a thorough empirical evaluation of our approach. Lastly, we discuss some closely related approaches (Sec. 5.4).

5.1 Zero-Shot Option Inventors

Our overall approach for solving long-horizon, stochastic robot planning problems is to zero-shot invent a set of options for the given problem (Alg. 4) and then to carry out hierarchical planning using these options (Alg. 5). In this section, we outline our approach for automatically identifying options (OptionInventor, Alg. 4) for a given environment.

Given a stochastic motion planning problem (Def. 1), Alg. 4 creates a zero-shot state abstraction (lines 1-5) using the methods presented in Sec. 4.2. Fig. 5.1(a) and (b) show this process in an example environment. Once abstract states are constructed, we define abstract actions as options (line 6) with their initiation set in one abstract state and the termination set in a different abstract state (discussed in Sec. 5.1.1). These options (action abstractions) are independent of problem instances, i.e., they are constructed once per environment and robot and reused for different problems (pairs of initial and goal configurations). However,

we still need to learn policies for executing such options. As defined, option termination sets turn out to be insufficient for efficiency: they result in a sparse-reward setting, which makes it difficult to scale RL algorithms for policy learning. To address this limitation, lines 7-9 also create in zero-shot fashion (without collecting additional experience from the environment), an *option guide*: a dense pseudo-reward function for the invented options (discussed in Sec. 5.1.2).

5.1.1 Zero-Shot Option Endpoints

Given a set of zero-shot abstract states $\mathcal{S}$ created using the predicted critical regions for a new environment (Def. 7), a neighborhood function $\mathcal{V}$, and an abstraction function α, we define two types of options: (1) *centroid options* that take the robot from the centroid of one critical region to another, and (2) *interface options* that take the robot across an abstract state, i.e., from the boundary between s_i and s_j to the boundary between s_i and s_k. Both forms of options can be composed to solve long-horizon problems (this process is discussed in the next section).

First, we discuss centroid options. Intuitively, these options define abstract actions that transition between a pair of critical regions. Formally, they are defined as follows:

Definition 10 *Let $s_i \in \mathcal{S}$ be an abstract state in the RBVD Ψ with the corresponding critical region $\phi_i \in \Phi$. Let d be the Euclidean distance measure and let t define a threshold distance. Let c_i be the centroid of the critical region r_i. A **centroid region** of the critical region r_i with the centroid c_i is defined as a set of configurations: $\{x|x \in s_i \wedge d(x, c_i) < t\}$.*

We use this definition to define the endpoints for the centroid options as follows:

Definition 11 *Let $s_i, s_j \in \mathcal{S}$ be neighboring abstract states such that $\mathcal{V}(s_i, s_j) = 1$ in an RBVD Ψ constructed using the set of critical regions Φ. Let $\phi_i, \phi_j \in \Phi$ be the critical regions for the abstract states s_i and s_j and let c_i and c_j be their centroids regions. The*

endpoints for a centroid option *are defined as a pair* $\langle \mathcal{I}_{ij}, \beta_{ij} \rangle$ *such that* $\mathcal{I}_{ij} = c_i$ *and* $\beta_{ij} = c_j$.

Interface options serve as dual to centroid options. Rather than defining high-level actions that move from the "center" of one abstract state to the "center" of another, they define high-level actions for going across an abstract state, from one boundary to another. To formally define interface options, we first need to define "interface" regions between a pair of neighboring abstract states:

Definition 12 *Let* $s_i, s_j \in \mathcal{S}$ *be a pair of neighboring states such that* $\mathcal{V}(s_i, s_j) = 1$ *and* ϕ_i *and* ϕ_j *be their corresponding critical regions. Let* $d^c(x, \phi)$ *define the minimum Euclidean distance between configuration* $x \in \mathcal{X}$ *and some point in a region* $\phi \subset \mathcal{X}$. *Let* p *be a configuration such that* $d^c(p, \phi_i) = d^c(p, \phi_j)$ *that is,* p *is on the border of the Voronoi cells that define* s_i *and* s_j. *Given the Euclidean distance measure* d *and a threshold distance* t, *an* ***interface region*** *for a pair of neighboring states* (s_i, s_j) *is defined as a set* $\{x | (x \in s_i \vee x \in s_j) \wedge d(x, p) < t\}$.

We use this definition of interface regions to define endpoints for the interface options as follows:

Definition 13 *Let* $s_i, s_j, s_k \in \mathcal{S}$ *be abstract states such that* $\mathcal{V}(s_i, s_j) = 1$ *and* $\mathcal{V}(s_j, s_k) = 1$. *Let* $\hat{\phi}_{ij}$ *and* $\hat{\phi}_{jk}$ *be the interface regions for pairs of high-level states* (s_i, s_j) *and* (s_j, s_k). *The* ***endpoints for an interface option*** *are defined as a pair* $\langle \mathcal{I}_{o_{ijk}}, \beta_{o_{ijk}} \rangle$ *such that* $\mathcal{I}_{o_{ijk}} = \hat{\phi}_{ij}$ *and* $\beta_{o_{ijk}} = \hat{\phi}_{jk}$.

Recall that the RBVD Ψ induces a neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$. We can now utilize these definitions to define, in zero-shot fashion, the complete set of centroid options and interface options for a new environment. The set of centroid options is defined

as $\mathcal{O}_c = \{o_{ij} | \forall s_i, s_j \in \mathcal{S}, \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{I}_{ij} = c_i \wedge \beta_{ij} = c_j\}$, where c_i represents the centroid of the critical region r_i for the abstract state s_i.

Similarly, the set of interface options is defined as $\mathcal{O}_i = \{o_{ijk} | \forall s_i, s_j, s_k \in \mathcal{S}, \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{V}(s_j, s_k) = 1 \wedge \mathcal{I}_{ij} = \hat{\phi}_{ij} \wedge \beta_{ij} = \hat{\phi}_{jk}\}$, where $\hat{\phi}_{ij}$ represents an interface region for a pair of neighboring abstract states s_i and s_j. Fig. 5.1(c) shows an example of automatically invented centroid and interface options for the environment shown in Fig. 5.1(a). These options can be used for hierarchical planning and learning as shown in Fig. 5.1(d) (explained in Sec. 5.2).

5.1.2 Zero-Shot Option Guides

Given an option defined using the methods discussed above, we define an *option guide* as a dense pseudo-reward function. We will use the option guide to improve sample efficiency while learning policy for an option in sparse reward settings.

Intuitively, option guides are defined using conceptual envelopes around *deterministic* motion plans that can be computed relatively easily using existing methods. Formally, we define an *δ-clear motion plan* as a motion plan in which every configuration has an δ-neighborhood that is collision-free. With a slight abuse of the notation, we use the abstraction function with a set of low-level configurations rather than a single configuration such that for a set A, $\alpha(A) = \{\alpha(x) | \forall x \in A\}$.

Let o_i be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$, and centroids $c_{\mathcal{I}_i}$ and c_{β_i} for $\mathcal{I}_i$ and β_i respectively.

Given a threshold distance t, an arbitrary neighborhood radius ϵ, and a Euclidean distance measure d, an ϵ-clear motion plan $\mathcal{G}$ for an option o is defined as $\mathcal{G} = \langle p_0, \ldots, p_n \rangle$ such that $p_0 = c_{\mathcal{I}}$, $p_n = c_\beta$, every point in $p_i \in \mathcal{G}$ has an ϵ-clear neighborhood, and for every pair of points $p_i, p_{i+1} \in \mathcal{G}$, $d(p_i, p_{i+1}) < t (< 2\epsilon)$. In practice, we found that any sampling-based motion planner with ϵ-inflated obstacles can be used to construct such motion plans.

We define the option guide for o_i as a dense pseudo-reward defined using $\mathcal{G}_i$ as follows. Intuitively, the option guide is a dense pseudo-reward function that provides the robot with a large positive reward when it reaches the termination set of the goal, a penalty for drifting to a different abstract state, and a smoothened reward for making progress on the option guide. Formally, this is defined as follows:

Definition 14 *Let o_i be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$ and let $\mathcal{G}_i = \langle p_1, \ldots, p_m \rangle$ be an ϵ-clear motion plan for a given ϵ. Given a configuration $x \in \mathcal{X}$, let $n(x) = p_i$ define the closest point on $\mathcal{G}_i$. The* ***option guide*** *$R_i(x)$ is defined as:*

$$R_i(x) = \begin{cases} r_t & \text{if } x \in \beta_i \\ r_p & \text{if } \alpha(x) \in \mathcal{S} \setminus \{\alpha(\mathcal{I}_i), \alpha(\beta_i)\} \\ -(d(x, n(x)) + d(n(x), p_m)) & \text{otherwise.} \end{cases}$$

The next section uses these concepts to present our approach to solving a stochastic motion planning problem.

5.2 Hierarchical Stochastic Motion Planning Using Zero-Shot Options

The SHARP algorithm (Alg. 5) presents our overall approach for using the zero-shot options defined above for hierarchical motion planning under uncertainty. It takes as input an SMP problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$, a simulator, and an occupancy matrix of the environment, and produces a partial policy $\Pi : \mathcal{X} \rightarrow \mathcal{U}$ that maps each reachable robot configuration to a control action. The algorithm starts by invoking the OptionInventor in line 2 to construct zero-shot state and action abstractions (in the form of options) if they have not been constructed for the given robot R and the environment E_{test} pair (Sec. 10).

Lines 4-19 use these options as high-level actions for computing high-level plans. Line 5 uses an incremental plan generator that takes the set of invented options along with the abstract initial and goal states as input and generates a high-level plan using A* search.

Algorithm 5: Stochastic Hierarchical Abstraction-guided Robot Planner (SHARP)

```
   Input: Training environments E_train, test environment E_test, initial and goal
          configurations x_i and x_g
   Output: A policy Π composed of options
1  if abstraction is not constructed then
2      𝒪, C ← OptionInventor(R, E_train, E_test);
3  s_i, s_g ← get_abstract_states(x_i, x_g);
4  while not refined do
5      p ← get_new_high_level_plan(s_i, s_g, 𝒪, C);
6      if p = ∅ then
7          break;
8      Π = empty_list;
9      π_0 ← lear_ll_policy(x_i, ℐ_{o_1});
10     Π.add(π_0);
11     foreach o ∈ p do
12         if π_o does not exist then
13             if 𝒢_o = ∅ then
14                 flag o infeasible;
15                 break;
16             π_o ← learn_ll_policy(ℐ_o, β_o, 𝒢_o);
17             adjust the option cost C_o;
18         Π.add(π_o);
19     refined ← True;
20 if refined then
21     π_{n+1} ← learn_ll_policy(β_{o_n}, x_g);
22     Π.add(π_{n+1});
23     return Π;
24 else
25     return failure;
```

This module considers the initiation and termination sets of the invented options as preconditions and effects. It uses the Euclidean distance between the termination set of the option and the goal configuration as the heuristic and the Euclidean distance between the initiation and termination sets as an initial approximation to the cost of the option.

Once a plan in the form of a sequence of options is obtained in line 5, SHARP starts refining each option in the plan by computing option policies. However, before computing the policy for the first option in the plan, it generates an additional option o_0 such that

$\mathcal{I}_{o_0} = x_i$ and $\beta_{o_0} = \mathcal{I}_{o_1}$ and learns its policy (line 9). If a policy exists for the option from the previous invocation of the algorithm, then our approach uses the same policy. Before computing a policy for an option, Alg. 5 checks for its option guide. If an option guide does not exist, the option is marked as infeasible and a new high-level plan is computed from the initial abstract state (line 14). Once an option guide is computed for an option, line 16 uses an off-the-shelf low-level policy learner to compute a policy for it. After computing (or reusing) policies for all the options in the plan, line 21 generates an additional option o_{n+1} such that $\mathcal{I}_{o_{n+1}} = \beta_{o_n}$ and $\beta_{o_{n+1}} = x_g$ and learns its policy.

Finally, we compute a *composed policy* by composing policies for every option in this high-level plan (lines 18 and 22). A **composed policy**, Π, for a high-level plan is a finite state controller with one state for each option in the plan. For a controller state q_i, $\Pi(x) = \pi_i(x)$ where π_i represents the policy for option $o_i \in \mathcal{O}$. The controller makes a transition $q_i \rightarrow q_{i+1}$ when the robot reaches a configuration $x \in \mathcal{I}_{o_{i+1}}$.

To aid transferability, SHARP only synthesizes options once per environment and robot. It efficiently transfers the learned option policies by updating the option costs (C) using the average number of steps from initiation sets to the termination sets of the options in multiple rollouts of the learned option policies (line 17).

5.2.1 Theoretical Results

Recall that a controller implicitly defines a transition function with a probability distribution $\mu(x + u)$ for the control action u (see Ch. 2). Then, a *δ-compliant controller* is defined as one whose set of support for $\mu(x + u)$ is $B_\delta(x + u)$. Here, we refer to δ as the *support radius* for the given controller. We now present the theoretical properties of Alg. 5. Let $B_\delta(x)$ for $\delta > 0$ define the δ-neighborhood of $x \in \mathcal{X}$ under the Euclidean metric. Our formal guarantees do not require knowledge of μ other than an upper bound on the support radius.

The following theoretical results characterize the formal properties of the presented approach. The main result (Thm. 5.2.1) shows that the construction process of the options ensures that the zero-shot options are indeed composable and can be used for high-level deterministic planning.

Theorem 5.2.1 *For a given stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_1, x_n \rangle$, let Φ be the set of identified critical regions and Ψ be the RBVD that induces the set of abstract state $\mathcal{S}$ and a neighborhood function $\mathcal{V}$. If there exists a sequence of distinct abstract states $\langle s_1, \ldots, s_n \rangle$ such that $\mathcal{V}(s_i, s_{i+1}) = 1$ then there exists a composed policy Π such that the resulting configuration after the termination of every option in Π is the goal configuration x_n.*

Thm. 5.2.3 asserts that when used with an optimal low-level policy learner, SHARP is probabilistically complete for holonomic robots.

Lemma 5.2.1 *Let $\mathcal{X}$ be the configuration space of the robot R and let Φ and Ψ be the set of critical regions and RBVD respectively inducing the set of abstract states $\mathcal{S}$ and the neighborhood function $\mathcal{V}$. If there exists a pair of neighboring abstract states $s_i, s_j \in \mathcal{S}$ such that $\mathcal{V}(s_i, s_j) = 1$ then there exists a pair of option endpoints $\mathcal{I}_{ij}$ and β_{ij} such that $\mathcal{I}_{ij} \subset s_i$ and $\beta_{ij} \subset s_j$.*

Proof 5.2.1 *The proof is straightforward and directly follows from the Alg. 5 itself. Our approach for creating options considers all pairs of neighboring abstract states and creates options that transition between them. For more details, refer to Sec. 10.*

Recall the definition of a δ-clear motion plan from Sec. 5.1.2. Then we have:

Proposition 5.2.1 *Let R be a holonomic robot using a δ_c-compliant controller. For an option o with a pair of endpoints $\langle \mathcal{I}_o, \beta_o \rangle$, if there exists an option guide between $\mathcal{I}_o$ and*

β_o in the form of a δ-clear motion plan such that $\delta_c < \delta$ then there exists a proper partial policy for the option o.

Proof 5.2.2 *Let $\mathcal{G}_o = \langle p_1, \ldots, p_n \rangle$ be an option guide for the option o as defined in Sec. 5.1.2. Here, each $p_i \in \mathcal{G}_o$ refers to a collision-free configuration $x_i \in \mathcal{X}_{free}$ that has a collision-free δ-neighborhood represented with $B_\delta(p_i)$. Now, given that the robot uses a δ_c-compliant controller such that $\delta_c < \delta$, an optimal partial proper policy can be defined using a function that gives the next closest point on the option guide moving towards the termination set of the option o. Let $N_o : \mathcal{X} \to \mathcal{X} : x \mapsto p_i$ such that $\forall j > i$, $d(p_j, x) > d(p_i, x)$ and $d(p_i, \beta_o) < d(x, \beta_o)$. An optimal policy is such that $\pi_o(x) = N_o(x)$ given a δ-clear $\mathcal{G}_i$. Given that the robot is using δ_c-compliant controller with the support radius $\delta_c < \delta$, the robot would always end up in the B_{δ_c} neighborhood of a point in the option guide which is a subset of B_δ collision-free neighborhood. This ensures the existence of a proper policy for the option o.*

Lemma 5.2.2 *Let R be a holonomic robot using a δ_c-compliant controller. If there exists a pair of option endpoints $\mathcal{I}_i$ and β_i with an option guide $\mathcal{G}_i$ in the form of a δ-clear motion plan between $\mathcal{I}_i$ and β_i such that $\delta_c < \delta$, and if the low-level policy learner is optimal, then Alg. 5 will learn an option $o_i = \langle \mathcal{I}_i, \beta_i, \mathcal{G}_i, \pi_i \rangle$.*

Proof 5.2.3 *The proof is straightforward. Proposition 5.2.1 proves existence of a proper policy π_i for an option with endpoints $\mathcal{I}_i, \beta_i$ and a holonomic robot R using δ_c-compliant controller if there exists δ-clear option guide $\mathcal{G}_i$ such that $\delta_c < \delta$. The rest of the proof relies on the optimality of the low-level learning. The option guide $\mathcal{G}_i$ also induces a dense pseudo-reward function R_i (Sec. 5.1.2) that provides a smooth reward function that guides the robot to the termination set. Given that the π_i is an optimal policy (proposition 5.2.1) and the low-level policy learner is optimal, it should compute π_i.*

Theorem 5.2.2 *For a given stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_1, x_n \rangle$, let Φ be the set of identified critical regions and Ψ be the RBVD that induces the set of abstract state $\mathcal{S}$ and a neighborhood function $\mathcal{V}$. If there exists a sequence of distinct abstract states $\langle s_1, \dots, s_n \rangle$ such that $\mathcal{V}(s_i, s_{i+1}) = 1$ then there exists a composed policy Π such that the resulting configuration after the termination of every option in Π is the goal configuration x_n.*

Proof 5.2.4 *The proof directly derives from the definition of the endpoints for the centroid and interface options. Given a sequence of adjacent abstract states $\langle s_1, \dots, s_n \rangle$, Def. 11 and 13 ensures a sequence of options $\langle o_1, \dots, o_n \rangle$ such that $\beta_i = \mathcal{I}_{i+1}$. This implies that an option can be executed once the previous option is terminated. Given this sequence of options $\langle o_1, \dots, o_n \rangle$, according to the definition of the composed policy, there exists a composed policy Π such that for every pair of consecutive options $o_i, o_j \in \Pi$, $\mathcal{I}_{o_j} = \beta_{o_i}$. Thus, we can say that if every option in Π terminates, then the resulting configuration would be the goal configuration.*

Theorem 5.2.3 *Given a stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$ for a holonomic robot R using a controller with a support radius $\delta_c < \delta$, a motion planner that can compute δ-clear motion plans, and an optimal low-level policy learner, if there exists a δ-clear motion plan for the robot R from x_1 to x_n that forms a sequence of distinct abstract states, then Alg. 5 will find a proper policy for the given stochastic motion planning problem.*

Proof 5.2.5 *Let $T = \langle x_i, \dots, x_g \rangle$ be the δ-clear motion plan from the initial configuration x_i to goal configuration x_g. This δ-clear motion plan forms a non-repeating sequence of abstract states. Let $p = \langle s_1, \dots, s_n \rangle$ be this sequence of distinct abstract states. Given that Alg. 5 explores all possible sequences of high-level states between a given pair of initial and goal abstract states (line 10), we can say that eventually, it would find this sequence of*

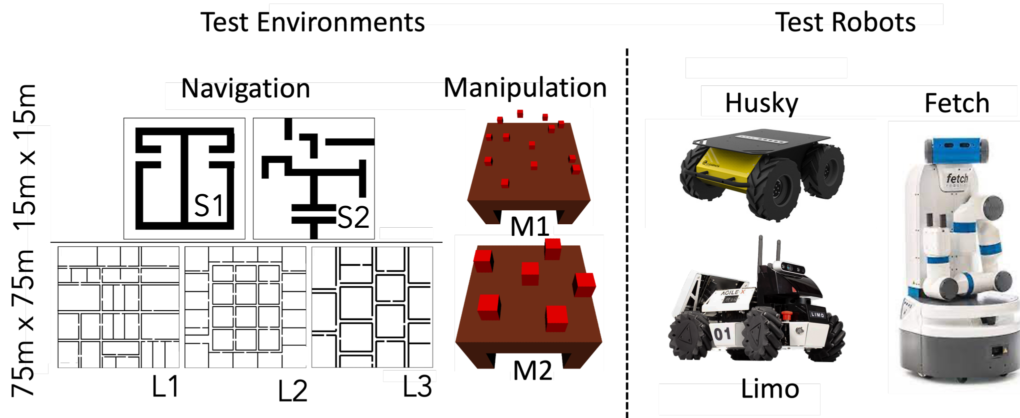


Figure 5.2: Our test environments and robots.

abstract states p *and the corresponding sequence of options for it. We can also deduce that for every pair of consequent abstract states* $s_j, s_{j+1} \in p$*, there exists a pair of consequent configurations* $x_j, x_{j+1} \in T$ *such that* $x_j \in s_j$ *and* $x_{j+1} \in s_{j+1}$ *and* $\mathcal{V}(s_j, s_{j+1}) = 1$ *and that there exists a* δ*-clear motion plan between abstract states* s_j *and* s_{j+1}*. Now, lemmas 5.2.1 and 5.2.2 show that given a motion planner that computes a* δ*-clear motion plan and an optimal low-level policy learner, Alg. 5 would be able to learn options with proper policies for every pair of neighboring states. This implies that our approach would be able to learn options for each pair of consequent states in* p*. Lastly, Theorem 5.2.2 proves that if there exists a sequence of distinct abstract states then there exists a composed policy of learned options that when executed successfully in* x_i *terminates in* x_g *i.e., a solution for the given problem.*

These results provide the foundations for analyzing such approaches and show a completeness result for the presented approach. However, our approach generalizes beyond the sufficient (and not necessary) conditions used in the theorems above. In fact, our empirical evaluation (Sec. 5.3) is conducted on non-holonomic robots that violate the premises of these results. Furthermore, we use default controllers with unknown support radii.

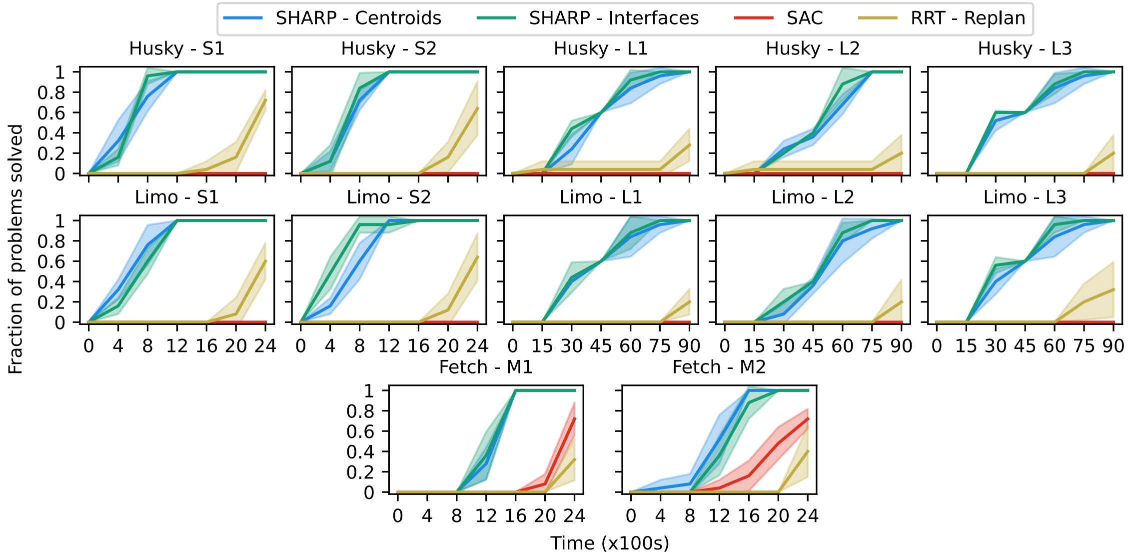


Figure 5.3: (Higher values are better) Times taken (averaged over 5 trials) by our approach (SHARP) and baselines to compute solutions in the test environments. The X-axis shows the time and the y-axis shows the fraction of the problems solved in the given time.

5.3 Empirical Results

We present the salient aspects of our implementation, setup, and observations here; additional results, code, and data are available in the supplementary material.

Our evaluation is organized to address the following key questions: (1) Does the presented approach of zero-shot option invention followed by hierarchical planning and refinement improve performance in terms of computational efficiency and solution quality?; and (2) Can zero-shot options be transferred to new problems in the same environment?

Results across an extensive evaluation suite indicate that the presented approach creates and uses zero-shot options effectively. In larger environments (L1-L3), ours is the only approach that shows significant learning, and it achieves a significantly higher solution quality than all baselines. We now present our evaluation framework and results in detail.

Evaluation framework and metrics We organized the overall evaluation of the presented approach as follows. Given a previously unseen environment E_{test} and a problem

instance $\langle x_i, x_g \rangle$, SHARP (Alg. 5) zero-shot invents options for E_{test} and uses them to compute a policy for the test problem instance. The total solution time recorded for SHARP includes the time taken to run OptionInventor (which includes predicting critical regions, creating state abstractions, inventing option signatures, and computing option guides), and to execute hierarchical planning and refinement process listed under the SHARP algorithm (Alg. 5).

We evaluated the *computational efficiency* of all considered approaches in terms of the number of problems solved in a given amount of time. For learning-based approaches, a problem is considered to be solved in these experiments when the current learned policy yields an average reward of +500 over 10 rollouts. For RRT-replan, a problem is considered to be solved when the robot reaches the 0.2m neighborhood of the goal configuration. All approaches were assigned a uniform timeout per problem of 2400 or 9000 seconds.

In addition, we use two metrics to evaluate solution quality since it is often easy to compute meaningless policies in a short time frame: The *average solution cost* is defined as the average number of steps taken while executing a computed solution; *solution reliability* is defined as the likelihood of solving the given problem by executing the computed solution. Both metrics are computed over 20 independent trials of the computed solution on the input problem instance.

Figs. 5.3 and 5.4 summarize the results of our evaluation in terms of these metrics across a wide range of robots, environments, and test problems. We discuss the details of this evaluation including notes on our implementation, environment, and baseline selection, and our main observations below.

Our implementation We implemented two variants of our approach: SHARP-centroids and SHARP-interfaces, which invent and use centroid options and interface options, respectively. Both implementations use PyBullet and PyTorch (Paszke *et al.*, 2019). PyBul-

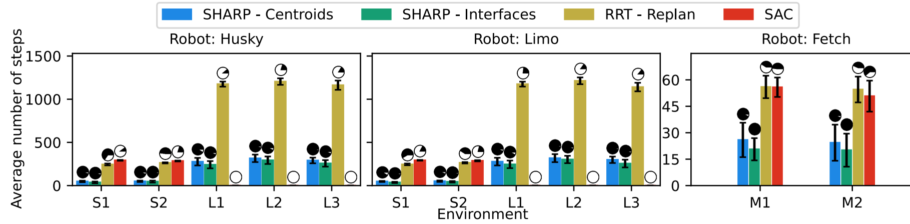


Figure 5.4: (Smaller bars and darker circles are better) An average number of steps taken in **successful** executions of the learned policies and success rates for our approach and the baselines. The pie chart over each bar represents the success rate (shaded black area) while executing the learned policy.

let does not feature stochasticity robot movements. We introduced stochasticity by adding random perturbations (unknown to Alg. 5) in control targets of actions during training and execution. We used default robot controllers to evaluate the learned policies. We used HARP (Shah and Srivastava, 2022b) with $\epsilon = 0$ for computing zero-shot option guides.

We used $2-$layered neural networks with 256 neurons in each layer for representing local policies for the learned options. Inputs to these networks were the current configuration of the robot and a vector to the nearest point on the option guide for the current option. We used +1000 as a pseudo reward for reaching the termination set of each option and -100 as a penalty for drifting to a different abstract state. We use SAC (Haarnoja *et al.*, 2018) as a low-level policy learner in lines 9, 16, and 21 of Alg. 5.

Test environments and robots We evaluated our approach across 7 test environments (Fig. 5.2) (not included in training the critical region predictor), 3 non-holonomic robots (Fig. 5.2) and a total of 60 navigation and manipulation problems. Dimensions of the environments are as follows: S1, S2: $15m \times 15m$; L1, L2, L3: $75m \times 75m$. Problem-specific timeouts were set at 2400s for small environments and manipulation problems and 9000s for larger environments. For each environment, we generated 5 problem instances by randomly sampling different initial and goal configuration pairs. We used the following robots: the ClearPath Husky (3-DOF), the AgileX Limo (3-DOF), and the Fetch manipulator robot

(7-DOF). The Husky is a 4-wheeled differential drive robot that can move in one direction and rotate in place; the Limo is also a 4-wheeled omnidirectional robot with an Ackermann dynamics; the Fetch is an 8-DOF manipulator robot.

Baseline selection We considered and evaluated several learning and planning approaches (LaValle, 1998; Haarnoja *et al.*, 2018; Kulkarni *et al.*, 2016; Lillicrap *et al.*, 2016; Levy *et al.*, 2019; Bagaria and Konidaris, 2020) as potential baselines for this work. Of these, only RRT-Replan (LaValle, 1998) and SAC (Haarnoja *et al.*, 2018) solved any problem instances within the timeouts discussed above. Therefore, we compared our approach against SAC and RRT-Replan. SAC is an off-policy deep reinforcement learning approach that learns a single policy for the overall stochastic motion planning problem. We used the same network architecture as ours for SAC's neural policy. We used a terminal reward of $+1000$ and a step reward of -1 to train the SAC agent. RRT-Replan is a version of the popular RRT algorithm that recomputes a plan from the robot's current configuration if the robot fails to successfully reach the goal after executing the initial plan. All approaches considered used the same input robot models, simulators, and low-level controllers as our approach.

5.3.1 Analysis of Results

Computational Efficiency Fig. 5.3 shows the fraction of problem instances solved in a given amount of time by both variants of SHARP and the baselines. In our case, this includes the time taken to create the abstract states and actions as well as to compute the solutions. Each subsequent problem uses learned high-level actions (policies and options) from the previous problem instances when available. Results show SHARP shows significantly greater scalability and computational efficiency. In most cases, baselines take $2\times$ the time taken by SHARP to compute a solution. These differences increase for larger environ-

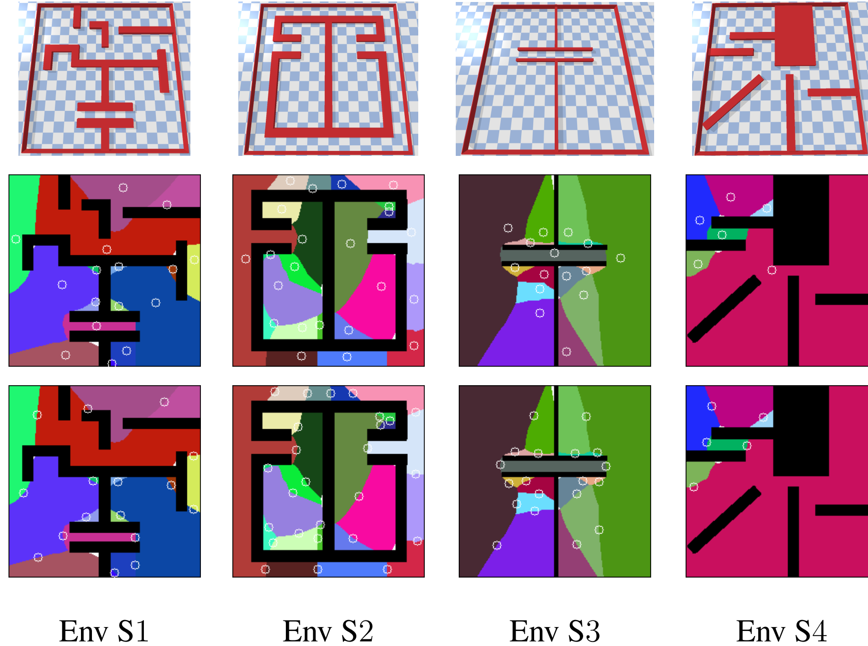


Figure 5.5: Test environments of the size $15m \times 15m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions used to synthesize centroid options. Bottom: The white circles represent the interface regions for each pair of abstract states used to synthesize interface options.ere

ments, where baselines were able to solve less than 50% of the environments that SHARP solved within the same timeouts.

These results illustrate the impact of learning to zero-shot invent and utilize options: even when the time for predicting critical regions, building abstractions, computing high-level plans, and learning low-level policies is included, SHARP significantly outperforms the baselines. Manipulation environments show a relatively smaller difference between the performance of all the approaches owing to smaller horizons. This reinforces the key contribution of our approach of creating problems with smaller horizons using options to solve problems with significantly large horizons.

Solution quality Fig. 5.4 shows solution cost and solution reliability (as defined above) for solutions computed by all considered approaches. These results show that SHARP's

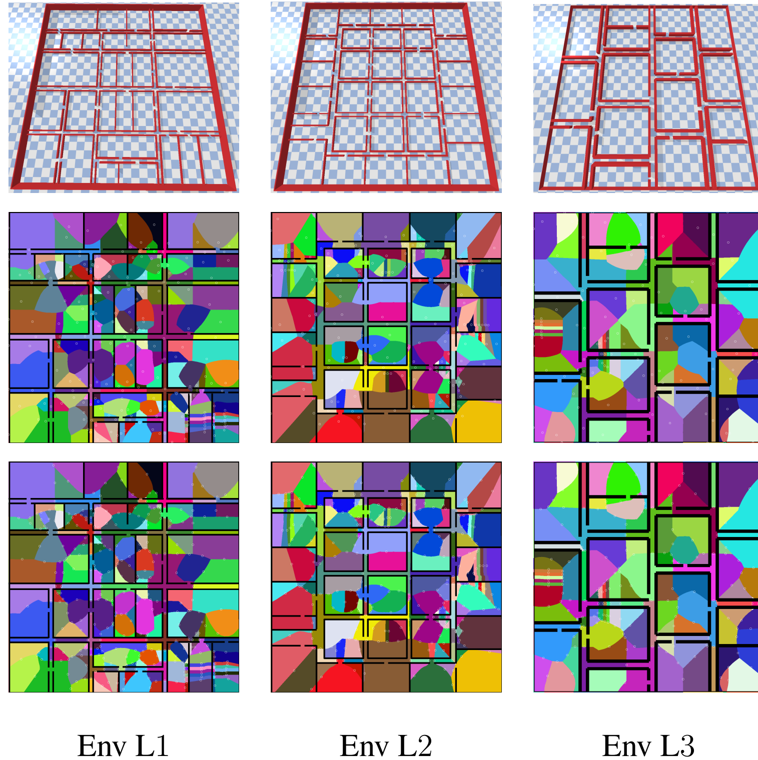


Figure 5.6: Test environments of the size $75m \times 75m$ with the identified abstract states. These images show 2D projections of high-dimensional region-based Voronoi diagrams. Each colored partition represents an abstract state. Top: The white circles represent centroids of the predicted critical regions used to synthesize centroid options. Bottom: The white circles represent the interface regions for each pair of abstract states used to synthesize interface options.

planning over zero-shot options results in lower cost solutions: they require significantly fewer steps during execution compared to baselines, with the differences frequently spanning *orders of magnitude*. We acknowledge that RRT-Replan is not an optimal planning approach. However, the solution quality also represents the amount of time RRT-Replan had to re-compute and re-execute the solution.

Computing policies that account for stochasticity makes SHARP's solution reliability uniformly above 90%, nearly $3\times$ that of RRT-Replan (the best-performing baseline) on the larger test environments. RRT-Replan's solutions had an execution success rate of ~50% in the smaller navigation (S1, S2) and manipulation (M1, M2) environments, and a success rate of less than 33% in the larger environments (L1-L3). SAC's solution reliability was

	S1	S2	L1	L2	L3	M1	M2
Interface Options	43%	33%	37%	33%	42%	50%	75%
Centroid Options	50%	50%	39%	36%	50%	65%	75%

Table 5.1: Percentage of options that our approach reused from the task they were computed to every subsequent task they were needed across 5 test tasks in each environment.

lower, indicating limited scalability of end-to-end learning in long-horizon problems.

Zero-shot option invention and reuse Fig. 5.5 and Fig. 5.6 show the predicted critical regions, 2D projections of the RBVDs, and synthesized option endpoints for our test environments. These results show that our approach is able to zero-short invent options for new, unseen test environments. When new problem instances come from a common environment, our approach is able to transfer these zero-shot options and their policies to new problem instances. Centroid options showed greater reuse rates on average across all environments (52%) than interface options (45%). Detailed re-use rates are available in Table 5.1.

Now, we discuss some of the closely related approaches.

5.4 Related Work

To the best of our knowledge, this is the first approach that uses a data-driven method for synthesizing transferable and composable options and leverages these options with a hierarchical algorithm to compute solutions for stochastic path planning problems. It builds upon the concepts of abstraction, stochastic motion planning, option discovery, and hierarchical reinforcement learning and combines reinforcement learning with planning. Here, we discuss related work in these areas.

Motion planning is a well-researched area. Numerous approaches (Kavraki *et al.*, 1996; LaValle, 1998; Kuffner and LaValle, 2000; Pivtoraiko *et al.*, 2009; Saxena *et al.*, 2022) have been developed for motion planning in deterministic environments. Kavraki *et al.* (1996); LaValle (1998); Kuffner and LaValle (2000) develop sampling-based techniques that randomly sample configurations in the environment and connect them for computing a motion plan from the initial and goal configurations. Holte *et al.* (1996); Pivtoraiko *et al.* (2009); Saxena *et al.* (2022) discretize the configuration space and use search techniques such as A* search to compute motion plans in the discrete space.

Stochastic motion planning Multiple approaches (Du *et al.*, 2010; Kurniawati *et al.*, 2012; Vitus *et al.*, 2012; Huynh *et al.*, 2016; Berg *et al.*, 2017; Hibbard *et al.*, 2022) have been developed for performing motion planning with stochastic dynamics. Alterovitz *et al.* (2007) build a weighted graph called stochastic motion roadmap (SMR) inspired by the probabilistic roadmaps (PRM) (Kavraki *et al.*, 1996) where the weights capture the probability of the robot making the corresponding transition. Huynh *et al.* (2016) extend SMR for computing stochastic policies through value iteration over motion trees constructed using RRT (LaValle, 1998). Sun *et al.* (2016) use linear quadratic regulator – a linear controller that does not explicitly avoid collisions – along with value iteration to compute a trajectory that maximizes the expected reward. However, these approaches require an analytical model of the transition probability of the robot's dynamics. Tamar *et al.* (2016) develop a fully differentiable neural module that approximates value iteration (VI) and can be used for computing solutions for stochastic path planning problems. However, these approaches (Alterovitz *et al.*, 2007; Sun *et al.*, 2016; Tamar *et al.*, 2016) require discretized actions. Du *et al.* (2010); Van Den Berg *et al.* (2012) formulate a stochastic motion planning problem as a POMDP to capture uncertainty in robot sensing and movements. Multiple approaches (Jurgenson and Tamar, 2019; Eysenbach *et al.*, 2019; Jurgenson *et al.*,

2020) design end-to-end reinforcement learning approaches for solving stochastic motion planning problems. These approaches only learn policies to solve one path-planning problem at a time and do not transfer knowledge across multiple problems. In contrast, our approach does not require discrete actions and it learns options that are transferrable to different problems.

Subgoal discovery Several approaches have considered the problem of learning task-specific subgoals. Kulkarni *et al.* (2016); Bacon *et al.* (2017); Nachum *et al.* (2018, 2019); Czechowski *et al.* (2021) use intrinsic reward functions to learn a two-level hierarchical policy. The high-level policy predicts a subgoal that the low-level goal-conditioned policy should achieve. The high-level and low-level policies are then trained simultaneously using simulations in the environment. Paul *et al.* (2019) combine imitation learning with reinforcement learning for identifying subgoals from expert trajectories and bootstrap reinforcement learning. Levy *et al.* (2019) learn a multi-level policy where each level learns subgoals for the policy at the lower level using Hindsight Experience Replay (HER) (Andrychowicz *et al.*, 2017) for control problems rather than long-horizon motion planning problems in deterministic settings. Kim *et al.* (2021) randomly sample subgoals in the environment and use a path planning algorithm to select the closest subgoal and learn a policy that achieves this subgoal.

Option discovery Numerous approaches (Stolle and Precup, 2002; Şimşek *et al.*, 2005; Brunskill and Li, 2014; Kurutach *et al.*, 2018; Eysenbach *et al.*, 2019; Bagaria and Konidaris, 2020; Bagaria *et al.*, 2021) perform hierarchical learning by identifying task-specific options through experience collected in the test environment and then use these options (Sutton *et al.*, 1999) along with low-level primitive actions. Stolle and Precup (2002); Şimşek *et al.* (2005) lay the foundation for discovering options in discrete settings. They collect

trajectories in the environment and use them to identify high-frequency states in the environment. These states are used as termination sets of the options and initiation sets are derived by selecting states that lead to these high-frequency states. Once options are identified, they use Q-learning to learn policies for these options independently to formulate Semi-MDPs (Sutton *et al.*, 1999). Bagaria and Konidaris (2020) learn options in a reverse fashion. They compute trajectories in the environment that reaches the goal state. In these trajectories, they use the last K points to define an option. These points are used to define the initiation set of the option and the goal state is used as a termination set. They continue to partition the rest of the collected trajectories similarly and generate a fixed number of options.

Several approaches (Watter *et al.*, 2015; Levine *et al.*, 2016; Finn *et al.*, 2016; Gal *et al.*, 2016; Henaff *et al.*, 2017; Tamar *et al.*, 2017; Ebert *et al.*, 2018; Amos *et al.*, 2018; Hafner *et al.*, 2019) have explored vision-based model predictive control for robot planning problems. These approaches learn latent representations of the kinematic and dynamics model of the robot and use them to perform model-based control for the given robot control problem. These approaches focus on stochastic optimal control problems. In contrast, our approach focuses on relatively long-horizon robot planning problems and can be used with arbitrary controllers for short-horizon control (~ 5 seconds).

Planning with options Approaches for combining symbolic planning with reinforcement learning (Silver and Ciosek, 2012; Yang *et al.*, 2018; Jinnai *et al.*, 2019; Lyu *et al.*, 2019; Kokel *et al.*, 2021; Konidaris *et al.*, 2018; Silver *et al.*, 2021) use pre-defined abstract models to combine symbolic planning with reinforcement learning. In contrast, our approach learns such options (including initiation and termination sets) as well as their policies and uses them to compute solutions for stochastic path planning problems with continuous state and action spaces.

Conclusion

This chapter presents the first approach that uses a data-driven process to learn to create state and action abstractions for unseen environments and problem instances. We provide theoretical results as well as a thorough empirical evaluation of the presented methods. These results show that the presented approach effectively learns to create abstractions that provide strong performance and quality advantages on a broad set of problems that are currently beyond the scope of known methods.

Chapter 6

AUTOMATICALLY INVENTING RELATIONAL WORLD MODELS FOR ROBOT PLANNING

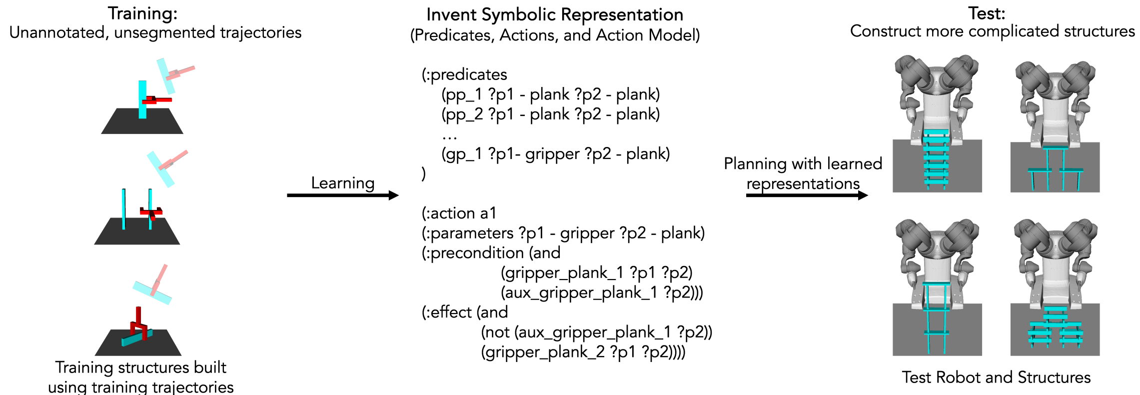


Figure 6.1: Our overall approach. We start with a set of demonstrations on relatively simple tasks using a simple robot and learn a symbolic model in the form of a set of predicates and high-level actions. This symbolic model can be used with any off-the-shelf planner for solving unseen complex long-horizon planning problems with other similar robots.

Solving complex robot planning problems requires robots to interact with different objects in the environment (Ch. 2). However, abstractions learned in Ch. 4 and Ch. 5 do not account for objects in the environment or changing configuration spaces. In this chapter, we present our novel approach for learning relational abstractions for robot planning problems that capture relations between robots and objects as well as different relations between objects.

Fig 6.1 provides a high-level summary of our approach. Our approach starts with a small set of training demonstrations and learns a symbolic world model that supports efficient planning. Here, a world model is represented in terms of state and action abstractions represented in PDDL comprised of symbolic predicates and actions defined using these predicates (Sec. 2.3). Our approach uses the concept of relative poses (described in

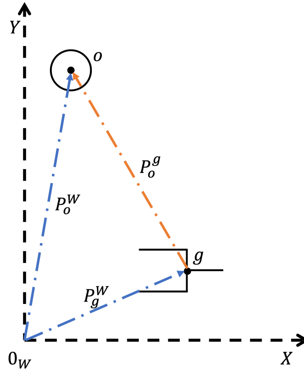


Figure 6.2: An illustration for computing relative poses

Sec. 6.1). Sec. 6.2 discusses our approach for automatically inventing predicates and high-level actions; Sec. 6.3 provides an algorithm for using invented abstraction for planning while continually updating them; Sec. 6.4 provides a thorough empirical evaluation of our approach; Sec. 6.5 discusses some of the related approaches.

6.1 Relative Poses

Our approach extensively uses *relative poses*. Every object in the environment also defines a frame of reference. A relative pose defines the pose of an object in the reference frame of another object. Basis transformations from linear algebra can be used to compute relative transformations of objects with reference to other objects in the environment.

Let o be an object in the environment and g be a gripper. Let P_o^W represent the pose of the object o in the world reference frame. Similarly, let P_g^W represent the pose of the gripper g in the world reference frame. These poses are also called absolute poses of the object o and gripper g.

Relative poses are defined between a pair of objects. The relative pose of an object defines a pose for the objects in the reference frame defined by another object. E.g., the relative pose of object o w.r.t. to gripper g defines the pose of the object relative to the gripper.

Concepts from the basis transformations from linear algebra can be used to compute these relative poses. In this case, we can re-write the equation for the absolute pose of the object o as follows,

$$P_o^W = P_g^W P_o^g$$

Using this, we can derive the following equation for the relative pose of the object o in the reference frame of the gripper g that only uses absolute poses of the objects as follows.

$$P_o^g = P_W^g P_o^W$$
$$P_o^g = \left(P_g^W\right)^{-1} P_o^W$$

We refer to the pose of an object o_1 relative to an object o_2 as $P_{o_1}^{o_2}$. Let $\tilde{\mathcal{X}}_{o_1}^{o_2}$ define a relative state-space for the pair of objects o_1 and o_2, i.e., the set of all poses of the object o_1 in the relative frame of the object o_2, and $\tilde{\mathcal{X}}$ define the set of relative state spaces such that $\tilde{\mathcal{X}} = \{\tilde{\mathcal{X}}_{o_j}^{o_i} | o_i, o_j \in \mathcal{O} \wedge o_i \neq o_j\}$. Lastly, we define a transformation function $\xi : \mathcal{X} \rightarrow \tilde{\mathcal{X}}$ that computes the relative state for each absolute state of the environment.

6.2 Learning World Models

This work's central idea is to automatically learn world models in the form of state and action abstractions that can be transferred to settings with different robots and goals. In this context, different goals are characterized by different numbers and configurations of the objects. We formally define the abstraction learning problem as follows.

Definition 15 *Let* T_{train} *be a set of training problems and* $\mathcal{D}_{\text{train}}$ *be a set of demonstrations that successfully solve the training problems. We define the* ***abstraction learning problem*** *as learning* 1) *a predicate vocabulary* $\mathcal{P}$*,* 2) *a set of high-level actions* $\bar{\mathcal{A}}$*, and* 3) *a set of generators* Γ *for each learned predicate.*

Algorithm 6: LAMP: Learning Abstract Model for Planning

Input: A set of demonstrations $\mathcal{D}_{train}$ for training tasks T_{train}, a set of objects $\mathcal{O}$, a set of types of objects $\mathcal{T}$, an initial state $x_i \in \mathcal{X}_{\text{free}}$, a goal state $x_g \in \mathcal{X}_{\text{free}}$, a motion planner MP, k

Output: PDDL Domain $\mathcal{M}$

```
/* Prepare data */
```
1 Use ξ to compute trajectories with relative poses of each object
```
/* Invent predicates */
```
2 Compute sets of relative critical regions Ψ for each pair of object types $\tau_i, \tau_j \in \mathcal{T}$

3 $i \leftarrow 0$;

4 $\Psi \leftarrow$ discover_new_goal_specific_relation(x_g, i);

5 $\mathcal{R} \leftarrow$ discover_relations(Ψ,$\mathcal{O}$, $\mathcal{T}$)

6 $\tilde{\mathcal{R}} \leftarrow$ discover_auxilary_relations($\mathcal{R}$);

7 $\mathcal{P} \leftarrow$ generate_predicate_vocabulary($\mathcal{R}$, Ψ)
```
/* Invent actions */
```
8 $\bar{\mathcal{A}} \leftarrow$ invent_actions($\mathcal{D}$, $\mathcal{P}$);

9 $\mathcal{M} \leftarrow$ generate_PDDL($\mathcal{T}$,$\mathcal{O}$,$\mathcal{R}$,$\bar{\mathcal{A}}$);

10 result $\leftarrow$ Plan(x_i, x_g, $\mathcal{M}$, MP, Γ, k);

11 **if** *result = failure & not all goal-specific relations are not added to* Ψ **then**

12   **go to** 4

13 **return** $\mathcal{M}$, *result*;

The core contribution of this work is the first known approach for simultaneously inventing a predicate vocabulary and abstract actions that solve unseen test problems T_{test} with similar robots and types of objects but significantly varying goals. We now present our approach – *Learning Abstract Model for Planning (**LAMP**)* – that automatically learns these abstractions in a continual fashion and represents them as a PDDL domain. The next section (Sec. 6.2.1) presents our approach for automatically inventing a predicate vocabulary and generators and then we present our approach for inventing symbolic actions (Sec. 6.2.2).

6.2.1 Inventing Predicates

We now discuss our approach for automatically discovering a predicate vocabulary only using the set of training demonstrations $\mathcal{D}_{train}$ for solving the set of training problems T_{train}.

Def. 6 define critical regions in the configuration space of a robot. However, these

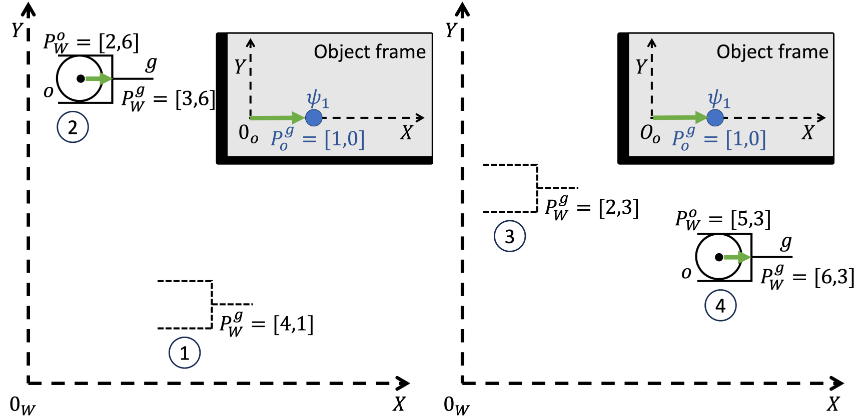


Figure 6.3: An example of relational critical regions. The gripper g is tasked to grasp the object o. Stages 1 and 3 show an initial configuration of the gripper g and stages 2 and 4 show the final configuration. The inset figures show the pose of the gripper for stages 2 and 4 in the relative reference frame of the object and the blue region shows the identified relative critical region.

critical regions fail to capture relationships between different objects in the environment. E.g., consider a simple task of grasping an object in a 2D setting with a gripper shown in Fig. 6.3. Here, every problem instance would have a different initial state and hence a different pose of the object o. This would require a different configuration of the gripper to grasp the object for every problem, and therefore the critical regions in the configuration space of the robot (as defined by Shah and Srivastava (2022b)) would fail to identify any useful abstractions.

Relational Critical Regions

In this work, we propose the concept of *relational critical regions* (RCR) that overcome shortcomings of critical regions. Relational critical regions extend the notion of critical regions to the relative spaces between two objects to capture salient relationships between objects that only exist in these spaces. E.g., in Fig. 6.3, the absolute poses of the gripper g and object o do not have a specific relationship, however, the relative poses of the object and the gripper display a "holding" relationship. The inset images in Fig 6.3 show the "holding"

relation between the object and the gripper and its corresponding relational critical region (blue regions in the inset images). Given a pair of objects o_1 and o_2 and the relative state space $\mathcal{X}_{o_2}^{o_1}$, relational critical regions can be defined similarly to critical regions (Def. 6) by considering the relative spaces between two objects ($\mathcal{X}_{o_2}^{o_1}$) instead of the configuration space of the robot ($\mathcal{X}_r$). Formally, given a criticality threshold υ, relational critical regions can be defined as follows.

Definition 16 *Let T be a robot planning problem and $\mathcal{D}_T$ be a set of solution trajectories for the planning problem T. Let $o_1, o_2 \in \mathcal{O}$ be a pair of objects and let $\mathcal{X}_{o_2}^{o_1}$ define the relative state space for object o_2 in the relative reference frame of the object o_1. The measure of criticality of a Lebesgue-measurable open set $\rho \subseteq \mathcal{X}_{o_2}^{o_1}$, $\mu(\rho)$, is defined as $\lim_{s_n \to^+ \rho} \frac{f(r)}{v(s_n)}$ where $f(\rho)$ is a fraction of observed solution trajectories solving for the planning problem T that contains a relative pose $P_{o_2}^{o_1}$ such that $P_{o_2}^{o_1} \in \rho$, $v(s_n)$ is the measure of s_n under a reference density (usually uniform), and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing ρ ($\rho \subseteq s_n$, $\forall\, n$).*

Now, we describe our approach for learning a set of relational critical regions.

Learning relational critical regions Alg. 6 describes our approach to learning symbolic abstractions. We start with the set of demonstrations $\mathcal{D}_{train}$ that solves the set of training problems T_{train}. Recall the function ξ defined in Sec. 6.1. Our approach uses the function ξ to convert training demonstrations $\mathcal{D}_{train}$ containing absolute poses of the objects and robot to relative demonstrations in relative state space $\tilde{\mathcal{X}}$ (line 1) and use these trajectories to identify relational critical regions (Def. 16).

Our approach assumes that objects of similar types interact similarly. E.g., the relational critical region between the object o and the gripper g generalizes to every similar object and gripper in the environment. Therefore, Alg. 6 accumulates demonstrations for similar

types of objects and then identifies critical regions between two types of objects. Alg. 6 first identifies task-specific relational critical regions and then combines them to construct the complete set of relational critical regions. Let Ψ be this set of automatically identified relational critical regions.

Inventing critical regions from the goal Our approach uses a limited set of demonstrations, and it may fail to capture relations for specific goals. Therefore, to invent goal-specific relational critical regions, we use the relative poses of the objects in the goal and update the existing set of relational critical regions Ψ with the new goal-specific regions. However, this can lead to exponentially many goal-specific critical regions. Therefore, we use an iterative approach that iteratively updates the set of critical regions with goal-specific regions based on their frequency in the goal. We start with the most frequent goal-specific region and use the decreasing order of the frequency for updating the set of relational critical regions.

Once a set of relational critical regions Ψ is constructed, our approach uses Gaussian parameters to parameterize the hypotheses space of the relational critical regions. Formally, let $\Psi_{ij} \subset \Psi$ be a set of relational critical regions between the pair of object types τ_i and τ_j. Given a pre-defined threshold ϵ, our approach uses Gaussian mixture model (GMM) to estimate Gaussian parameters μ_ψ and Σ_ψ for every relational critical region $\psi \in \Psi_{ij}$ such that support for every pose $P \in \psi$ is greater than epsilon, i.e., for every pose $P \in \psi$, for every relational critical region $\psi \in \Psi_{ij}$, support for every pose $P \in \psi$ is $\mathit{Pr}(P; \mathcal{N}(\mu_\psi, \Sigma_\psi)) > \epsilon$.

Now, we describe our approach to inventing relations using the learned relational critical regions.

Representing Invented Critical Regions as Relations

We use the identified relational critical regions to define a set of relations between objects in the environment. Let τ_i, τ_j be a pair of object types from the set of types $\mathcal{T}$ and let $\Psi_{ij} = \{\psi_1, \ldots, \psi_n\} \subset \Psi$ be the set of critical regions for the type of objects τ_i and τ_j. For each pair of object types $\tau_i, \tau_j \in \mathcal{T}$, we define a parameterized functional relation $r_{ij} : \mathcal{O}_{\tau_i} \times \mathcal{O}_{\tau_j} \times \Psi_{ij} \to \{T, F\}$. Given a pair of low-level objects o_i and o_j of object types τ_i and τ_j respectively and a relation critical region $\psi_k \in \Psi_{ij}$, a grounded relation $r_{ij}\,(o_i, o_j, \psi_k)$ is true in a low-level state $x \in \mathcal{X}$ if $P_{o_j}^{o_i} \in \psi_k$. Additionally, we define a relation such that for a given pair of objects o_i and o_j, $r_{ij}(o_i, o_j, \psi_0) \implies [\forall \psi_k \in \Psi_{ij}, \neg r_{ij}(o_i, o_j, \psi_k)]$. E.g., Fig. 6.3 shows low-level states where the relation $r_{og}(o, g, \psi_0)$ is true for configurations 1 and 3 and the relation $r_{og}(o, g, \psi_1)$ is true for configurations 2 and 3. Let R be the set of all relations between each pair of types of objects.

We define an auxiliary relation for each relation $r \in \mathcal{R}$ using a key geometrical property of the relational critical regions. Intuitively, this relation captures the number of objects that a relational critical region can occupy. Auxiliary relations are defined using the free volume of the critical region and the volume of the objects that are supposed to occupy the region. Formally, let $\psi_k \in \Psi_{ij}$ define a relational critical region for object types τ_i and τ_j. Let $\rho(\psi_k)$ (or $\rho(o_i)$) define the total volume of the region ψ_k (or object o_i) and let $\rho_{free}(\psi_k)$ define the free volume of the region ψ_k given a current state $x \in \mathcal{X}$. For every relation $r_{ij} \in \mathcal{R}$, we define an auxiliary relation $\tilde{r}_{ij} : \mathcal{O}_{\tau_i} \times \Psi_k \to \{0, 1\}$ such that given a pair of objects o_i and o_j of types τ_i and τ_j and a relational critical region $\psi_k \in \Psi_{ij}$, $\tilde{r}_{ij}(o_i, \psi_k) \implies \rho_{free}(\psi_k) > \rho(o_j)$. Let $\tilde{\mathcal{R}}$ be the set of these auxiliary relations.

A critical advantage of inventing relations in such a bottom-up fashion is that the pose generators (Sec. 2.3) do not have to be explicitly defined. Instead, automatically learned relational critical regions also serve as learned pose generators. This is explained in detail

Algorithm 7: Inventing Symbolic Actions

Input: Set of demonstrations $\mathcal{D}_{train}$, learned predicates $\mathcal{P}$
Output: Set of lifted actions $\bar{\mathcal{A}}$

1 $\bar{\mathcal{D}}'_{train} \leftarrow$ get_abstract_demonstrations($\mathcal{D}_{train}$,$\mathcal{P}$);
2 $\bar{\mathcal{D}}_{train} \leftarrow$ lift_demonstrations($\bar{\mathcal{D}}'_{train}$);
3 changed_predicates ← [];
4 **foreach** $d^k \in \bar{\mathcal{D}}$ **do**
5   **foreach** *consecutive state* $s_i, s_j \in d^k$ **do**
6         ${}^{+}C^k_{ij} \leftarrow s_j \setminus s_i$; ${}^{-}C^k_{ij} \leftarrow s_i \setminus s_j$;
7         $C^k_{ij} \leftarrow \langle {}^{+}C^k_{ij}, {}^{-}C^k_{ij} \rangle$;
8   changed_predicates.add(C^k_{ij});
9 $\mathcal{C} \leftarrow$ create_clusters($\bar{\mathcal{D}}_{train}$, changed_predicates);
10 $\bar{\mathcal{A}} \leftarrow$ [];
11 **foreach** $(S_i \rightarrow S_j), C \in \mathcal{C}$ **do**
12   *eff* ← ⟨*add* = ${}^{+}C$, *del* = ${}^{-}C$⟩;
13   *pre* ← $\cap_{s \in S_i} s$;
14   *pre* ←prune_precondition(*pre*);
15   *param* ← extract_params($S_i \rightarrow S_j$);
16     $\bar{\mathcal{A}}$.add(create_action(*param*, *pre*, *eff*));
17 **return** $\bar{\mathcal{A}}$

in Sec. 6.3.

Generating predicate vocabulary Relations invented by our approaches can be easily translated into PDDL representation (or any other representation). For a given pair of object types $\tau_i, \tau_j \in \mathcal{T}$, let $\Psi_{ij} \subset \Psi$ be the set of critical regions. For each relational critical region $\psi_k \in \Psi_{ij}$, each relation r_{ij} can be translated into a binary predicate $(\mathtt{p}^{\psi_k}_{\mathtt{ij}}\ ?\mathtt{y_i}\ ?\mathtt{y_j})$ where $?\mathtt{y_i}$ is a typed parameter of type τ_i and $?\mathtt{y_j}$ is a typed parameter of type τ_j. Similarly, each auxiliary predicate $\tilde{r}_{ij}$ can be translated to a unary predicate $(\tilde{\mathtt{p}}^{\psi_k}_{\mathtt{ij}}\ ?\mathtt{y_i})$ where $?\mathtt{y_i}$ is a typed parameter of type τ_i. Let $\mathcal{P}$ be a set of predicates for all relations (line 7).

Now, we describe our approach for inventing high-level actions using the identified predicates.

6.2.2 Inventing Symbolic Actions

In this work, we aim to learn relational actions that can be used efficiently for transfer and generalization. To achieve this, our approach invents high-level lifted actions, each of which corresponds to at least one change in the abstract state represented using the predicates discovered earlier (Sec. 6.2.1). Alg. 7 explains our approach for inventing high-level actions. This corresponds to line 6 in Alg. 6.

Identifying High-Level Actions

The first step in inventing high-level actions is to first identify these actions. In order to identify high-level actions, we first abstract every training demonstration in $d = \langle x_0, x_1, \ldots, x_n \rangle$, where $d \in \mathcal{D}_{train}$, to an abstract demonstration $\langle s'_0, s'_1, \ldots, s'_n \rangle$ using the invented predicates $\mathcal{P}$ such that $s'_i \in 2^{\mathcal{P}'}$ (line 1) and then to a lifted demonstration $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_i \in 2^{\mathcal{P}}$ (line 2). Given the set of abstract demonstrations $\bar{\mathcal{D}}_{train}$, line 7 computes a set of changed predicates C for each transition in every lifted demonstration in $\bar{\mathcal{D}}_{train}$. Precisely, for a demonstration $d^k \in \bar{\mathcal{D}}_{train}$ and a pair of consecutive lifted states $s_i, s_j \in d^k$, let ${}^{+}C^k_{ij} = s_j \setminus s_i$ and ${}^{-}C^k_{ij} = s_i \setminus s_j$ and let C^k_{ij} be $\langle {}^{+}C^k_{ij}, {}^{-}C^k_{ij} \rangle$. Line 9 uses these sets of changed predicates and cluster transitions such that each cluster has all the transitions corresponding to the same set of changed predicates. Let $\mathcal{C}$ denote these clusters where each cluster $c = \langle S_i \rightarrow S_j, C_{ij} \rangle$ is a tuple where each element is a set of transitions ($S_i \rightarrow S_j$) that correspond to the similar changed predicates C_{ij}. Each cluster $c_i \in \mathcal{C}$ induces a high-level action $\bar{a}_i \in \bar{\mathcal{A}}$. This approach is similar to Verma *et al.* (2022) and Silver *et al.* (2022). However, they use grounded states to identify actions while our approach uses lifted predicates (lines 10-15). Next, we discuss our approach for learning effects, preconditions, and parameters for each invented action and explain it in detail in with an example (Ex. 4).

Learning Symbolic Action Model

Once a set of high-level actions $\bar{\mathcal{A}}$ is identified, we use associative learning to learn a symbolic model for each automatically identified action $\bar{a} \in \bar{\mathcal{A}}$. A symbolic model for an action is represented in terms of its symbolic effects, symbolic preconditions, and action parameters. Our approach also learns the symbolic model for each high-level action using the set of training demonstrations $\mathcal{D}_{train}$ as follows.

Learning effects In our setting, effect of an action $\bar{a}$ is represented as $eff_{\bar{a}} = \langle add_{\bar{a}}, del_{\bar{a}} \rangle$ (Sec. 2.3). Each cluster $c_i \in \mathcal{C}$ is generated by clustering transitions in $\bar{\mathcal{D}}$ over the sets of changed predicates. These changed predicates correspond to added and removed predicates as an effect of executing the action induced by the cluster. Therefore, for an action $\bar{a}_i$ induced by the cluster c_i with a set of changed predicates $C_i = \langle {}^{+}C_i, {}^{-}C_i \rangle$, $add_{\bar{a}_i} = {}^{+}C_i$ and $del_{\bar{a}_i} = {}^{-}C_i$ (line 12).

Learning preconditions To learn the precondition of an action, we take the intersection of all states where the action is applicable. Given a possible set of predicates, this approach generates a maximal precondition that is conservative yet sound (Wang, 1994; Stern and Juba, 2017). To do this, given an action $\bar{a} \in \bar{\mathcal{A}}$ corresponding to a cluster $c = \langle S_i \rightarrow S_j, C_{ij} \rangle$, $pre_{\bar{a}} = \cap_{s \in S_i} s$ (line 13).

Each action can have spurious preconditions corresponding to static relations that do not change on applying the action but are still true in all the pre-states $s \in S_i$. Therefore, we remove predicates (line 14) from the learned precondition that (i) are not parameterized by any of the objects that are changed by the action, and (ii) are not changed at any point in any of the demonstrations. This removes any predicate from the precondition that is spurious with respect to the data.

Learning action parameters Once the precondition and effect of an action are learned, the final step is to learn the parameters of the action, that can be replaced with objects to ground the action. In this step, the predicates in precondition and effect are processed in order. These predicates are processed in alphanumeric order and each of their parameters is added to the action's parameter list, if not added already. This process leads to an ordered list of parameters of the action, which can be grounded with compatible objects (line 15).

Example 4 *Let the set of predicates invented in Sec. 6.2.1 be the following:*

- `(table-can0 ?table ?can)`*: The can is not on the table.*
- `(table-can1 ?table ?can)`*: The can is on the table.*
- `(can-gripper1 ?can ?gripper)`*: The gripper is at the grasp pose (not holding/grasping yet).*
- `(can-gripper2 ?can ?gripper)`*: The gripper has grasped the object.*
- `(base-gripper0 ?base ?gripper)`*: The Robot's base link and the robot's gripper link do not have any relation.*
- `(base-gripper1 ?base ?gripper)`*: The robot's arm is tucked so there is a specific relative pose between the robot's base link and the robot's gripper link.*
- `(base-table1 ?base ?table)`*: The robot's base link is located in a way such that the robot's arm can reach objects on the table.*

Now, consider the two trajectories T_1 and T_2 as shown in Fig. 6.4 and Fig. 6.5, respectively. Here T_1 corresponds to the Fetch robot picking a yellow cup, and T_2 corresponds to the robot picking up a green cup (kept at a different location on the table compared to that of the yellow cup). Here these two trajectories are expressed in terms of grounded objects.

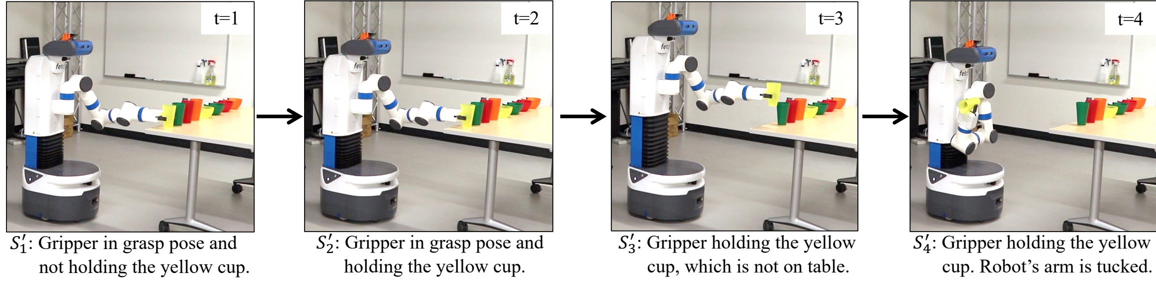


Figure 6.4: Trajectory $T_1 = \langle S_1', S_2', S_3', S_4' \rangle$ corresponding to the process of picking up a yellow cup from the table. The state description below each image explains that image in English. These state descriptions are added here for ease of understanding only.

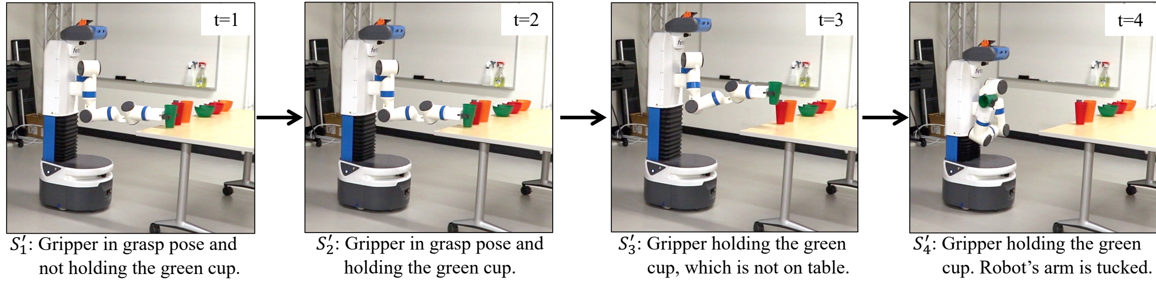


Figure 6.5: Trajectory $T_2 = \langle S_1', S_2', S_3', S_4' \rangle$ corresponding to the process of picking up a green cup from the table. The state description below each image explains that image in English. These state descriptions are added here for ease of understanding only.

These are converted to a lifted form using line 2 of Alg. 7 in terms of the predicates shown earlier. For T_1 *and* T_2 *both, the lifted states will be:*

- S_1 : { `(table-can1 ?table ?can)`, `(can-gripper1 ?can ?gripper)`, } { `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.
- S_2 : { `(table-can1 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, } { `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.
- S_3 : { `(table-can0 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, } { `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.
- S_4 : { `(table-can0 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, } { `(base-gripper1 ?base ?gripper)`, `(base-table1 ?base ?table)`}.

Note that we only show partial states here for brevity. The actual states will also have predicates like `(table-can1 ?table ?can2)`, `(table-can1 ?table ?can3)`, `(table-can1 ?table ?can4)`, `(table-can1 ?table ?bowl1)`, `(table-can1 ?table ?bowl2)`, `(table-can1 ?table ?bowl3)`, *etc. corresponding to other objects kept on the table.*

Learning effects *Alg. 7 creates the following three clusters (lines 4-9) based on these states.*

- $C_{12} = \langle {}^{+}C_{12} = \{$ `(can-gripper2 ?can ?gripper)` $\}, {}^{-}C_{12} = \{$ `(can-gripper1 ?can ?gripper)` $\}\rangle$.
- $C_{23} = \langle {}^{+}C_{23} = \{$ `(table-can0 ?table ?can)` $\}, {}^{-}C_{23} = \{$ `(table-can1 ?table ?can)` $\}\rangle$.
- $C_{34} = \langle {}^{+}C_{34} = \{$ `(base-gripper1 ?base ?gripper)` $\}, {}^{-}C_{34} = \{$ `(base-gripper0 ?base ?gripper)` $\}\rangle$.

Learning preconditions *Learning preconditions involve taking the intersection of states in which all the actions in the same cluster were executed. Here S_1 to S_3 mentioned below will remain the same for the three clusters. For e.g., precondition of $C_{12} = \{$* `(table-can1 ?table ?can)`, `(can-gripper1 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)` $\}$. *Alg. 7 will prune out* `(base-table1 ?base ?table)` *from the precondition as (i) it is unchanged between S_1 and S_2, and (ii) none of its parameters (*`?base` *and* `?table`*) are part of any other predicate that is changed. Using this, the precondition for each action will be:*

- pre$(C_{12}) = \{$ `(table-can1 ?table ?can)`, `(can-gripper1 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)` $\}$.

- pre(C_{23}) = { `(table-can1 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.

- pre(C_{34}) = { `(table-can0 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.

Learning parameters *Learning parameters from an action's precondition and effect is straightforward. All the unique parameters in predicates in the precondition and effect are added to the parameter list of an action representing a cluster. Using this notion, the parameters for the three clusters will be the following:*

- param(C_{12}) = `(?table ?can ?gripper ?base)`.
- param(C_{23}) = `(?table ?can ?gripper ?base)`.
- param(C_{34}) = `(?table ?can ?gripper ?base)`.

Now, we describe our approach for using the learned abstractions with any off-the-shelf task and motion planner while continually learning new relations and actions.

6.3 Planning with Learned Abstractions and Continual Learning

This section discusses our approach to using the symbolic model learned using Alg. 8 for solving new unseen long-horizon planning problems. Planning (Alg. 8) starts with the learned symbolic model $\mathcal{M} = \langle \mathcal{T}, \mathcal{P}, \bar{\mathcal{A}} \rangle$, a set of learned generators Γ, a motion planner MP, an initial state $x_i \in \mathcal{X}_{\text{free}}$, and a set of goal states $\mathcal{X}_g \subseteq \mathcal{X}_{\text{free}}$. Line 1 uses the set of learned predicates $\mathcal{P}$ to compute the symbolic initial state s_i.

After computing the abstract initial state, line 2 utilizes the set of goal states $\mathcal{X}_g$ and the learned predicate vocabulary $\mathcal{P}$ to create a relation graph G corresponding to the specified goal states. A relation graph is a directed graph with objects in the environment as nodes

```
Algorithm 8: Planning with Learned Model
  Input: an initial state x_i ∈ X_free, a goal state x_g ∈ X_free, learned symbolic model M,
         a motion planner MP, a set of generators Γ, k, training demonstrations D_train
  Output: A plan of primitive actions π, updated model M
1  s_i ← get_abstract_state(x_i);
2  G ← create_relation_graph(x_g,P);
3  update_model ← False;
4  π ← [] ;
5  R_missing ← {};
6  A_inaccurate ← {};
7  while solution not found or G has no edges do
8      s_g ← create_symbolic_goal(G);
9      Π̄ ← compute_k_symbolic_plans(M, O, s_i, s_g, k);
10     if Π̄ = ∅ then
11         p ← relax the relation graph G;
12         P_missing.add(p);
13         update_model ← True;
14         continue;
15     foreach π̄ ∈ Π̄ do
16         foreach ā ∈ π̄ do
17             a ← refine_action(ā, Γ);
18             if refinement fails then
19                 A_inaccurate.add(ā);
20                 update_model ← True;
21                 goto to next symbolic plan;
22             π.append(a);
23     if update_model then
24         P_new ← P ∪ P_missing;
25         generate a trajectory D_π using π;
26         M ← learn_actions(D ∪ {D_π}, P_new);
27     return π, M
28 return failure, M
```

and predicates between objects as edges. Given the relation graph, lines 8 and 9 generate a PDDL goal and use a top-k-planner to compute a set of distinct high-level plans $\bar{\Pi}$. Once high-level plans are generated, any off-the-shelf task and motion planner can be used to refine one of these plans and generate a sequence of primitive actions π.

Learning pose generators Typically, a task and motion planner requires pose generators to be provided as input. However, because Alg. 6 invents predicates in a bottom-up manner using relational critical regions, the relational critical regions can serve as sampling-based pose generators for the learned predicates. Given a predicate $(\mathtt{p}_{\mathtt{ij}}^{\psi_{\mathtt{k}}} ?\mathtt{y}_{\mathtt{i}} ?\mathtt{y}_{\mathtt{j}})$ defined using a relation between object types τ_i and τ_j with a relational critical region $\psi_k \in \Psi_{ij}$, a pose generator Γ_r can be implemented as a sampler that samples a relative pose P from the distribution $\mathcal{N}(\mu_{\psi_k}, \Sigma_{\psi_k})$. A pose P is a valid sample iff $\mathit{Pr}(P; \mathcal{N}(\mu_{\psi_k}, \Sigma_{\psi_k})) > \epsilon$. Here each pose generator defines a relative pose. The grounded generator for a given low-level state $x \in \mathcal{X}$. It can be computed using concepts of basis transformations as outlined in Sec. 6.1

Updating abstractions Our approach relies on associative learning from passively collected data to invent symbolic predicates and actions, and to learn the action models. Therefore, it is possible that learned abstractions are incorrect. Our approach uses continual learning for continuously updating the set of predicates, the set of actions, and the action model.

One potential issue arises if the invented actions are insufficient to achieve the goal, leading the top-k-planner to fail in computing any high-level plan (see line 10). In this case, Alg. 8 relaxes the relation graph G by removing an edge from the relation graph at random. It uses the relaxed relation graph to generate a new goal and compute high-level solutions for it. Every time Alg. 8 relaxes the relation graph G by removing an edge from the graph, it stores the corresponding predicate in the set of missing predicates $\mathcal{P}_{\mathit{missing}}$ (line 12). These predicates are used to update the symbolic model. This process is repeated until at least one high-level plan is found or the relation graph G does not have any edges.

An inaccurate action model can also cause task and motion planning failures leading to unrefineable high-level plans. If refinement of an action fails, Alg. 8 flags the action as

inaccurate (line 19) and moves to the next symbolic plan to find a refineable high-level plan from the set of high-level plans $\bar{\Pi}$ (line 21).

Finally, if Alg. 8 had failed to find high-level plans without relaxing the relation graph or computing refinement for an action, we update the symbolic model $\mathcal{M}$ using the predicates in the set of missing predicates $\mathcal{P}_{missing}$. Let $\mathcal{D}_\pi$ be the trajectory induced by the solution π. To update the model, Alg. 8 simply re-invents the actions and re-learns the actions models using predicates $\mathcal{P} \cup \mathcal{P}_{missing}$ and training demonstrations $\mathcal{D}_{train} \cup \{\mathcal{D}_\pi\}$.

We now present a thorough evaluation of our approach in various settings with different robots.

6.4 Empirical Evaluation

We present the salient aspects of our implementation, setup, and observations here. Our empirical evaluation is designed to answer the following key questions: 1) Are the learned abstractions sound and generalizable to unseen complex planning problems?; 2) Are the learned abstractions transferable to different *similar robots* – robots with similar geometries for end-effectors?; and 3) How close the learned abstractions are to human intuition?

Results across various environments show that the presented approach learns powerful abstractions that are effective in solving new unseen problems that are far more complex than the demonstrations used to learn these abstractions. We now present our evaluation framework and results in detail.

Evaluation framework We evaluate our approach as follows. Given an environment E, we use a set of training demonstrations for learning a symbolic model $\mathcal{M}_E = \langle \mathcal{T}, \mathcal{P}, \bar{\mathcal{A}} \rangle$ if the model is not already learned. Once a symbolic model is learned, we evaluate the model using a set of test problems where each problem is defined as a pair $\langle \mathcal{O}, x_i, \mathcal{X}_g \rangle$ where $\mathcal{O}$

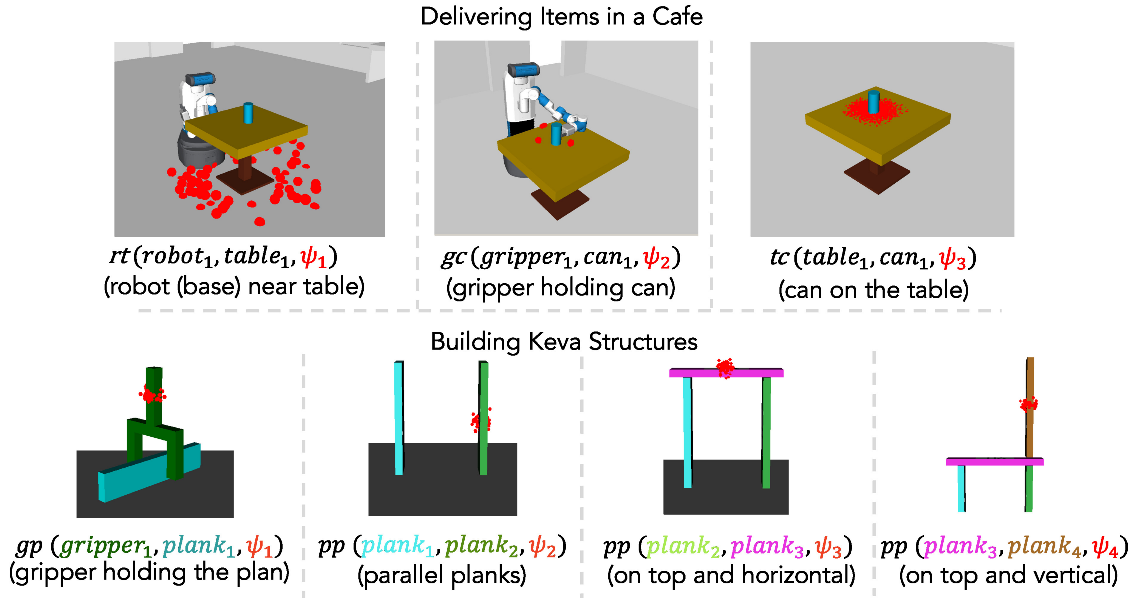


Figure 6.6: Different relations invented by our approach and their corresponding critical regions. Each image shows one binary predicate and its semantic interpretation. The red dots show sampled possible poses for the object in the relational critical region.

is the set of objects, x_i is an initial state and $\mathcal{X}_g$ is a set of goal states such that each test problem can have a different number of objects, different initial poses of the objects, and/or different target poses of the object.

We measure the generalizability of our domain by evaluating the success rate of solving unseen test problems using the learned model. We consider a test successful when Alg. 8 successfully computes a sequence of low-level actions using the learned symbolic model. In a subset of test environments (detailed later), we use different robots to learn the symbolic model and evaluate it to test the transferability of the learned abstract model. This allows us to use simpler robots at the time of training to generate demonstrations. Lastly, we also evaluate the semantic interpretation of the learned model by carrying out a manual analysis of learned predicates and actions. We now discuss the test environments and robots used to evaluate our approach.

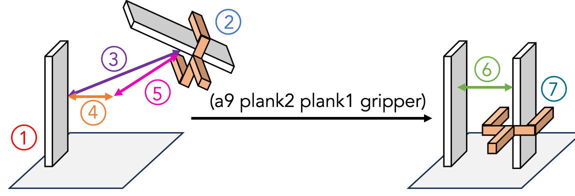
Test environments and robots We evaluate our approach in the following different environments.

(i) *Building Keva structures (Keva):* The first environment uses a robot and Keva planks to construct 3D structures. The robot can pick and place the planks to construct these 3D structures. Building these complex structures requires long-horizon planning with a large number of objects and actions that achieve various configurations of the planks. Appendix A.1 shows the structures used to learn the model and evaluate our approach. We use two different robots to learn and evaluate the abstractions in this environment showing the transferability of the learned abstractions. We use a simple disembodied gripper to learn symbolic abstractions and use the ABB YuMi robot with a 7-DOF arm in test tasks to evaluate the leaned abstractions.

(ii) *Delivering items in a cafe (CafeWorld):* A fetch robot is tasked to deliver items to different tables. The fetch robot is a mobile manipulator with an omnidirectional base and an 8-DOF arm. We use this environment to show the effectiveness of our learned abstractions in mobile manipulation tasks where different actions involve moving the robot base or its arm.

(iii) *Packing cans in a box (Packing):* This is a popular task and motion planning test environment where a robot is tasked to pack multiple objects in a small box. We use a disembodied gripper for this environment that works as a suspended robot in a factory picking objects from the top.

Baseline selection This is the first known approach that automatically invents symbolic predicate vocabularies, symbolic actions, and models for high-level planning directly from raw demonstrations. Therefore, there are no suitable baselines that input the same information and generate the same output. Nevertheless, we compare our approach with




```
Grounding: ?plank_p2 = plank2 ?plank_p1 = plank1 ?gripper_p1 = gripper
```

```
(:action a9
:parameters ( ?plank_p2 - plank ?plank_p1 -
              plank ?gripper_p1 - gripper )
:precondition (and
   (not (= ?plank_p2 ?plank_p1))
   (goalLoc_plank_1 goalLoc_Const ?plank_p1) ①
   (gripper_plank_1 ?gripper_p1 ?plank_p2)   ②
   (plank_plank_0 ?plank_p1 ?plank_p2)       ③
   (plank_plank_0 ?plank_p2 ?plank_p1)
   (aux3_plank_plank_1 ?plank_p1)            ④
   (goalLoc_plank_0 goalLoc_Const            ⑤
   ?plank_p2))
```

```
:effect (and
 (plank_plank_1 ?plank_p1 ?plank_p2)                    ⑥
 (not (aux3_plank_plank_1 ?plank_p1))
 (not (plank_plank_0 ?plank_p1 ?plank_p2))
 (goalLoc_plank_1 goalLoc_Const ?plank_p2)              ⑦
 (not (goalLoc_plank_0 goalLoc_Const ?plank_p2))))
```

Figure 6.7: An automatically invented high-level action by our approach. The top figure shows the states before and after executing the high-level action. The bottom part shows the automatically learned precondition and effect of the high-level action.

Code-as-policies (CoP) (Liang *et al.*, 2023). CoP takes input the high-level actions that the robot can execute and Python code snippets to execute these actions and uses a pre-trained LLM to compute a high-level plan. We also compare our approach with the oracle abstractions generated by an expert used with an off-the-shelf task and motion planner (Srivastava *et al.*, 2014). We set a timeout of 3600 seconds for our approach and baselines to compute high-level plans and refine them into a sequence of primitive actions.

We now discuss the analysis of the results on our test setup.

6.4.1 Analysis of Results

We analyze the invented abstractions for the following properties: Interpretability, scalability, effectiveness, robustness, and transferability.

Interpretability The core contribution of this work is autonomously learning symbolic abstractions for robot planning problems. However, for these abstractions to be useful, they need to be meaningful to a human. Therefore, we carefully examine the invented predicates

as well as high-level actions.

Notably, our approach autonomously invented meaningful relations despite the absence of annotations or labels in the training demonstrations, demonstrating its capability to derive semantic interpretations automatically. This does not only make abstractions invented by our approach effective, but also makes them interpretable. Fig. 6.6 illustrates a subset of invented relations by our approach. These predicates include predicates invented in the initial model using training demonstrations shown in Fig. 6.1 as well as All these predicates are invented while solving the set of test problems with an initial model constructed using training demonstrations shown in Fig. 6.1. Red regions depict approximations of the learned relational critical regions for corresponding objects using sampled poses. It can be seen from Fig. 6.6 that our approach autonomously captures crucial relations between objects in terms of the invented predicates. E.g., our approach automatically invents predicates that represent robot near the table and gripper at the grasp pose of the object in the mobile manipulation domain CafeWorld, and it invents relations between different planks such as parallel, on top vertically, and on top horizontally in the Keva domain.

Our approach also learns meaningful and human interpretable actions. E.g., Fig. 6.7 shows one of the automatically invented high-level actions. On careful examination, we can say that it corresponds to a high-level action that places a plank parallel to an already placed plank. More specifically, when grounded with the objects in the figure, the grounded action `(a9 plank2 plank1 gripper)` is placing `plank2` parallel to `plank1` using the `gripper`. Fig. 6.7 also shows preconditions and effects for the automatically invented "place parallel" action. The learned preconditions include: (i) `plank1` should have been placed, (ii) `gripper` should be holding `plank2`, (iii) `plank1` and `plank2` should not be parallel, (iv) no plank should be parallel to `plank1`, and (v) `plank2` should not be already placed. Similarly, the effects include relations corresponding to (i) `plank1` and `plank2` are parallel and (ii) `plank2` is placed. This highlights the ability of our approach

to learn effective, yet, intuitive high-level actions. We provide full auto-generated PDDL domains for all test domains in the appendix.

Domain	$\lvert\mathcal{D}_{train}\rvert$	$\lvert\mathcal{O}_{train}\rvert$	$\lvert\mathcal{O}_{test}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\bar{\mathcal{A}}\rvert$	**Used** $\lvert\mathcal{P}\rvert$	**Used** $\lvert\bar{\mathcal{A}}\rvert$	**Success rate**	**Avg. plan length**	**Avg. planning time (seconds)**	**Avg. refinement time (seconds)**
CafeWorld	500	1	3-8	22	12	21	11	1.0	74	0.17	658.23
Keva	50	1-2	3-24	24	12	17	8	1.0	50	1.92	92.89
Packing	50	1	2-4	8	5	8	5	0.96	20	0.11	476.82

Table 6.1: Detailed statistics about the empirical evaluation and invented abstractions. The success rate is an average of 50 independent test tasks with 5 random seeds.

Scalability Table 6.1 presents key observations for our empirical evaluation. It reports the number of training trajectories ($|\mathcal{D}_{train}|$), number of objects in the training demonstrations ($|\mathcal{O}_{train}|$), and number of objects in the test tasks ($|\mathcal{O}_{train}|$), number of invented predicates ($|\mathcal{P}|$) and actions ($|\bar{\mathcal{A}}|$), number of predicates and actions that were used in one of the test problems, number of objects in the training demonstrations ($|\mathcal{O}_{train}|$), and number of objects in the test tasks, success rate averaged over 50 randomly generated test tasks, and average plan length in test tasks. We also report average times (in seconds) needed to compute high-level plans using off-the-shelf planner task planner (Speck *et al.*, 2020) and times taken by an off-the-shelf task and motion planner (Srivastava *et al.*, 2014) to refine a high-level plan.

It is evident from table 6.1 that our approach can invent effective abstractions from a few demonstrations that generalize to significantly difficult test problems with significantly high numbers of objects as well as large branching factor and long horizons. E.g., our approach was able to automatically invent abstractions using only 50 demonstrations (including ~ 50% random demonstrations that do not achieve the goal) that included not more than 2 planks and a gripper and use it to compute successful solutions for test tasks than contained up to 24 planks, highlighting scalability of the invented abstractions.

Domain	$\|\mathcal{D}_{train}\|$	Solver						
		LAMP (our approach)					Code as Policies	TAMP
		20%	40%	60%	80%	100%		
CafeWorld	500	0.00 ± 0.00	0.98 ± 0.04	0.98 ± 0.04	0.98 ± 0.04	1.00 ± 0.00	0.00 ± 0.00	1.0 ± 0.0
Keva	50	0.00 ± 0.00	0.92 ± 0.00	0.95 ± 0.04	0.95 ± 0.04	1.00 ± 0.00	0.00 ± 0.00	1.0 ± 0.0
Packing	50	0.10 ± 0.11	0.92 ± 0.04	0.96 ± 0.05	0.92 ± 0.08	0.90 ± 0.13	0.00 ± 0.00	0.95 ± 0.07

Table 6.2: Ablation study of our approach with decreased training demonstrations and comparison with baseline approaches. The success rate for our approach and baselines averaged over 10 different unseen test tasks and 5 random executions. The percentages represent the fraction of training demonstrations used for learning the initial state and action abstractions.

Effectiveness In Table 6.2, we present the percentage of successfully solved test tasks using abstractions learned by our method, alongside the performance of two recent approaches that use expert-crafted abstractions as discussed above. These values are averaged across 10 diverse test tasks and through 5 random executions of our approach. Notably, the test tasks are more demanding than the training tasks, each involving at least three times the number of objects compared to any task in the training set used for learning initial symbolic abstractions. Snippets of the execution of solutions computed by LAMP (our approach) for various test tasks are provided in Appendix A.1. We can see from Table 6.2 that our approach significantly outperformed CoP and performed as good as the TAMP oracle. CoP used human-crafted high-level actions as well as needed manual effort to resolve syntactical errors in the output code. Yet, it failed to solve a single task from the set of test tasks across every domain.

Robustness Table 6.2 also illustrates the number of training demonstrations utilized for different evaluations of our approach, emphasizing the scalability and generalizability of our method even with limited data. With a modest number of training demonstrations, our approach outperformed the baselines in complex problems. It successfully tackled most tasks with abstractions learned in a few-shot manner. Specifically, our approach achieved a 100% success rate in tasks involving building structures with Keva planks and packing

cans in a box, needing only 50 trajectories. This underscores the data-efficiency of our approach and its ability to generalize effectively from a small number of demonstrations. Despite the non-trivial nature of learning abstractions for a mobile manipulation problem, our method efficiently solved 100% of these tasks in a cafe setting. However, for more intricate trajectories, such as grasping cans from different sides while positioning the robot on various sides of the table, our approach required a relatively higher number of trajectories (500). These training demonstrations, as noted above, also include ~ 50% randomly demonstrations that do not achieve the goal.

Transferability Our approach invents abstractions in a portable fashion. The invented abstractions can also be transferred between different robots. To evaluate the transferability of the invented abstractions, we use different robots to learn the abstractions and evaluate them. For the tasks involving building structures with Keva planks, we use a disembodied gripper as a robot in the training demonstrations. Fig. 6.1 and Fig. 6.6 show the disembodied gripper used to learn the abstractions. However, all the test tasks were solved using an ABB YuMi robot with a constrained 7-DOF arm, shown in Fig. 6.1. This underscores our approach's ability to invent abstractions that can be transferred between robots with different kinematic constraints but similar geometries.

6.5 Related Work

The presented approach directly relates to various concepts in task and motion planning, model learning, and abstraction learning. However, to the best of our knowledge, this is the first work that automatically invents generalizable symbolic predicates and high-level actions simultaneously using a set of low-level trajectories.

Task and motion planning approaches (Srivastava *et al.*, 2014; Dantam *et al.*, 2018; Garrett *et al.*, 2020; Shah *et al.*, 2020) develop approaches for autonomously solving long-

horizon robot planning problems. These approaches are complementary to the presented approach as they focus on using provided abstractions for efficiently solving the robot planning problems. Shah and Srivastava (2022b, 2024) learn state and action abstractions for long-horizon motion planning problems. An orthogonal research direction (Mishra *et al.*, 2023; Cheng *et al.*, 2023; Fang *et al.*, 2023) learns implicit abstractions (low-level generators or high-level skills) for task and motion planning in the form of generative models. However, these approaches do not learn generalizable relational representations as well as complex high-level relations and actions which is the focus of our work.

Several approaches invent symbolic vocabularies given a set of high-level actions (or skills) (Konidaris *et al.*, 2014; Ugur and Piater, 2015; Konidaris *et al.*, 2015; Andersen and Konidaris, 2017; Konidaris *et al.*, 2018; Bonet and Geffner, 2019; James *et al.*, 2020). Ahmetoglu *et al.* (2022); Asai *et al.* (2022); Liang and Boularias (2023) learn symbolic predicates in the form of latent spaces of deep neural networks and use them for high-level symbolic planning. However, these approaches assume high-level actions to be provided as input. On the other hand, the approach presented in this work automatically learns high-level actions along with symbolic predicates.

Numerous approaches (Yang *et al.*, 2007; Cresswell *et al.*, 2009; Zhuo and Kambhampati, 2013; Aineto *et al.*, 2019; Verma *et al.*, 2021) have focused on learning preconditions and effects for high-level actions, i.e., action model. A few approaches (Čertický, 2014; Lamanna *et al.*, 2021) have also focused on continually learning action models while collecting experience in the environment. Bryce *et al.* (2016) and Nayyar *et al.* (2022) focus on updating a known model using inconsistent observations. However, these approaches require a set of symbolic predicates and/or high-level action signatures as input whereas our approach automatically invents these predicates and actions. Several approaches (Silver *et al.*, 2021; Verma *et al.*, 2022; Chitnis *et al.*, 2022; Silver *et al.*, 2022; Kumar *et al.*, 2023; Silver *et al.*, 2023) have been able to automatically invent high-level actions that are

induced by state abstraction akin to the presented approach. However, unlike our approach, these approaches do not automatically learn symbolic predicates and/or low-level samplers and require them as input.

LLMs for robot planning Recent years have also seen significantly increased interest in using foundational models such as LLM (large language model), VLM (visual language model), and transformers for robot planning and control owing to their success in other fields such as NLP, text generation, and vision. Several approaches (Brohan *et al.*, 2022; Goyal *et al.*, 2023; Shridhar *et al.*, 2023; Vuong *et al.*, 2023) use transformer architecture for learning reactive policies for short-horizon robot control problems. Problems tackled by these approaches are analogous to individual actions learned by our approach.

Several directions of research explore the use of LLMs as high-level planners to generate sequences comprising of high-level, expert-crafted actions (Yu *et al.*, 2023; Liang *et al.*, 2023; Huang *et al.*, 2022; Rana *et al.*, 2023; Lin *et al.*, 2023; Huang *et al.*, 2023b; Ahn *et al.*, 2023). These methods make progress on the problem of near-natural language communication with robots and are complementary to the proposed work. However, there is strong evidence against the soundness of LLMs as planners. Valmeekam *et al.* (2023) show that LLMs are only ~ 36% accurate as planners even in simple block stacking settings not involving more than 5 object.

On the other hand, approaches that utilize LLMs to translate user requirements to formal specifications (Yu *et al.*, 2023; Ding *et al.*, 2023; Liu *et al.*, 2023b,a; Kwon *et al.*, 2023; Huang *et al.*, 2023a) are complimentary to our approach. These approaches input a set of symbolic predicates and use LLMs for automatically generating symbolic goals from natural language specifications. These goals can be further used by existing planners.

Conclusion

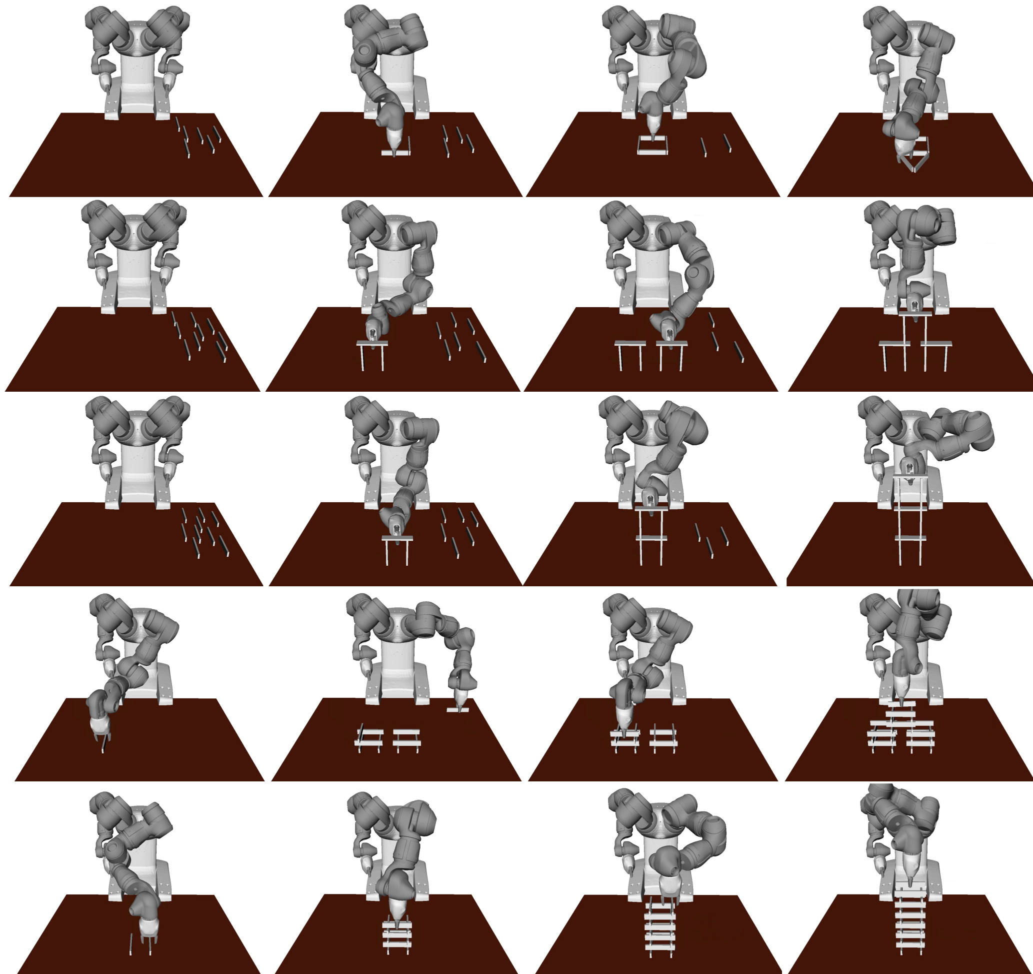
This chapter presents the first known approach for using the unsegmented and unannotated continuous low-level demonstration to invent symbolic state and action abstractions that generalize to different robots and unseen problem settings. Thorough evaluation in simulated and real-world settings shows that the learned abstractions are efficient and sound, as well as generate comprehensible abstractions.

APPENDIX

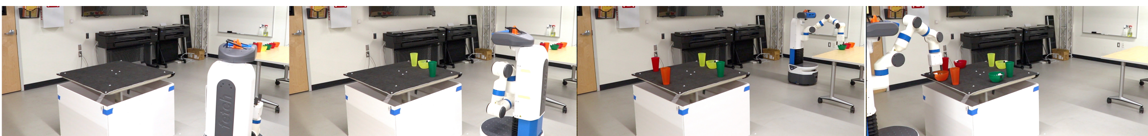
A.1 Test Environments

We show snippets of some of our different simulated and real-world experiments.

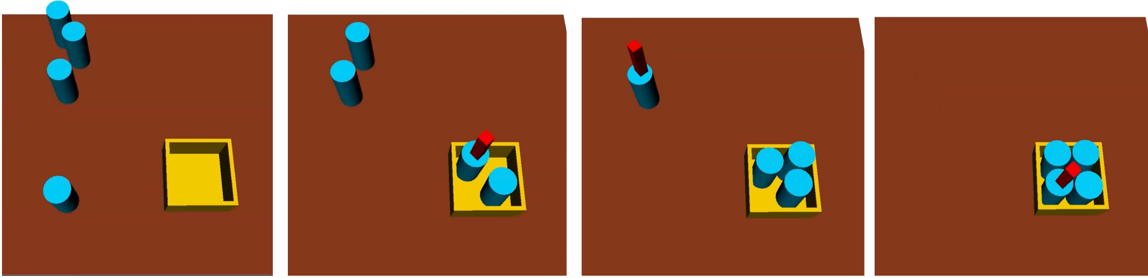
i) Building structures with Keva planks using a total of 20 random training demonstrations.



ii) Delivering items in a cafe



iii) Packing cans in a box



A.2: Learned PDDL Domains

Domain: Keva

```
(define (domain Keva)
(:requirements :strips :typing :equality :conditional-effects
    :existential-preconditions :universal-preconditions)
(:types
    goalLoc
    plank
    gripper
)

(:constants
    goalLoc_Const - goalLoc
 )

(:predicates
    (gripper_plank_0 ?x - gripper ?y - plank)
    (gripper_plank_1 ?x - gripper ?y - plank)
```

```
    (gripper_plank_2 ?x - gripper ?y - plank)
    (gripper_plank_3 ?x - gripper ?y - plank)
    (gripper_plank_4 ?x - gripper ?y - plank)
    (plank_plank_0 ?x - plank ?y - plank)
    (plank_plank_1 ?x - plank ?y - plank)
    (goalLoc_plank_0 ?x - goalLoc ?y - plank)
    (goalLoc_plank_1 ?x - goalLoc ?y - plank)
    (aux3_gripper_plank_0 ?x - gripper)
    (aux3_gripper_plank_1 ?x - gripper)
    (aux3_plank_plank_1 ?x - plank)
    (aux3_gripper_plank_2 ?x - gripper)
    (aux3_gripper_plank_3 ?x - gripper)
    (aux3_gripper_plank_4 ?x - gripper)
)

(:action a1
:parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
:precondition (and
    (gripper_plank_2 ?gripper_p1 ?plank_p1)
    (aux3_gripper_plank_1 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
)
:effect (and
    (gripper_plank_1 ?gripper_p1 ?plank_p1)
    (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
```

```
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
    (aux3_gripper_plank_2 ?gripper_p1)
    (not (aux3_gripper_plank_1 ?gripper_p1))
 )
)

(:action a2
:parameters ( ?gripper_extra_p1 - gripper  ?plank_p1 - plank )
:precondition (and
    (gripper_plank_1 ?gripper_extra_p1 ?plank_p1)
    (goalLoc_plank_0 goalLoc_Const ?plank_p1)
)
:effect (and
    (goalLoc_plank_1 goalLoc_Const ?plank_p1)
    (not (goalLoc_plank_0 goalLoc_Const ?plank_p1))
 )
)

(:action a3
:parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
:precondition (and
    (gripper_plank_0 ?gripper_p1 ?plank_p1)
    (aux3_gripper_plank_1 ?gripper_p1)
    (aux3_gripper_plank_2 ?gripper_p1)
```

```
    (aux3_gripper_plank_3 ?gripper_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
)
:effect (and
    (gripper_plank_2 ?gripper_p1 ?plank_p1)
    (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (aux3_gripper_plank_0 ?gripper_p1)
    (not (aux3_gripper_plank_2 ?gripper_p1))
 )
)

(:action a4
:parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
:precondition (and
    (goalLoc_plank_1 goalLoc_Const ?plank_p1)
    (gripper_plank_2 ?gripper_p1 ?plank_p1)
    (aux3_gripper_plank_0 ?gripper_p1)
    (aux3_gripper_plank_1 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
)
:effect (and
    (gripper_plank_0 ?gripper_p1 ?plank_p1)
```

```
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
    (aux3_gripper_plank_2 ?gripper_p1)
    (not (aux3_gripper_plank_0 ?gripper_p1))
 )
)

(:action a5
:parameters ( ?plank_p2 - plank  ?plank_p1 - plank  ?
        gripper_p1 - gripper )
:precondition (and
    (not (= ?plank_p2 ?plank_p1))
    (plank_plank_0 ?plank_p2 ?plank_p1)
    (gripper_plank_4 ?gripper_p1 ?plank_p1)
    (gripper_plank_2 ?gripper_p1 ?plank_p2)
    (plank_plank_1 ?plank_p1 ?plank_p2)
    (goalLoc_plank_1 goalLoc_Const ?plank_p2)
    (goalLoc_plank_1 goalLoc_Const ?plank_p1)
    (aux3_gripper_plank_0 ?gripper_p1)
    (aux3_gripper_plank_1 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
)
:effect (and
    (gripper_plank_0 ?gripper_p1 ?plank_p1)
```

```
    (gripper_plank_0 ?gripper_p1 ?plank_p2)
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_2 ?gripper_p1 ?plank_p2))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p2))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p2))
    (not (gripper_plank_3 ?gripper_p1 ?plank_p2))
    (aux3_gripper_plank_2 ?gripper_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
    (not (aux3_gripper_plank_0 ?gripper_p1))
 )
)

(:action a6
:parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
:precondition (and
    (goalLoc_plank_1 goalLoc_Const ?plank_p1)
    (gripper_plank_2 ?gripper_p1 ?plank_p1)
    (aux3_gripper_plank_1 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
)
:effect (and
    (gripper_plank_0 ?gripper_p1 ?plank_p1)
```

```
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
    (aux3_gripper_plank_2 ?gripper_p1)
 )
)

(:action a7
:parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
:precondition (and
    (gripper_plank_1 ?gripper_p1 ?plank_p1)
    (goalLoc_plank_1 goalLoc_Const ?plank_p1)
    (aux3_gripper_plank_2 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
)
:effect (and
    (gripper_plank_2 ?gripper_p1 ?plank_p1)
    (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (aux3_gripper_plank_1 ?gripper_p1)
    (not (aux3_gripper_plank_2 ?gripper_p1))
 )
)
```

```
(:action a8
:parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
:precondition (and
    (gripper_plank_0 ?gripper_p1 ?plank_p1)
    (aux3_gripper_plank_1 ?gripper_p1)
    (aux3_gripper_plank_2 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
)
:effect (and
    (gripper_plank_2 ?gripper_p1 ?plank_p1)
    (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_4 ?gripper_p1 ?plank_p1))
    (not (aux3_gripper_plank_2 ?gripper_p1))
 )
)

(:action a9
:parameters ( ?plank_p2 - plank  ?plank_p1 - plank
        ?gripper_p1 - gripper )
:precondition (and
    (not (= ?plank_p2 ?plank_p1))
    (plank_plank_0 ?plank_p1 ?plank_p2)
```

```
    (plank_plank_0 ?plank_p2 ?plank_p1)
    (gripper_plank_0 ?gripper_p1 ?plank_p1)
    (goalLoc_plank_0 goalLoc_Const ?plank_p2)
    (gripper_plank_1 ?gripper_p1 ?plank_p2)
    (goalLoc_plank_1 goalLoc_Const ?plank_p1)
    (aux3_gripper_plank_2 ?gripper_p1)
    (aux3_gripper_plank_3 ?gripper_p1)
    (aux3_plank_plank_1 ?plank_p2)
    (aux3_plank_plank_1 ?plank_p1)
    (aux3_gripper_plank_4 ?gripper_p1)
)
:effect (and
    (gripper_plank_4 ?gripper_p1 ?plank_p1)
    (goalLoc_plank_1 goalLoc_Const ?plank_p2)
    (plank_plank_1 ?plank_p1 ?plank_p2)
    (not (gripper_plank_3 ?gripper_p1 ?plank_p1))
    (not (plank_plank_0 ?plank_p1 ?plank_p2))
    (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
    (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
    (not (goalLoc_plank_0 goalLoc_Const ?plank_p2))
    (aux3_gripper_plank_0 ?gripper_p1)
    (not (aux3_gripper_plank_4 ?gripper_p1))
    (not (aux3_plank_plank_1 ?plank_p1))
 )
)
```

```
)
```

Domain: CafeWorld

```
(define (domain CafeWorld)
(:requirements :strips :typing :equality :conditional-effects
    :existential-preconditions :universal-preconditions)
(:types
    goalLoc
    freight
    can
    gripper
    surface
)

(:constants
    goalLoc_Const - goalLoc
 )

(:predicates
    (gripper_can_0 ?x - gripper ?y - can)
    (gripper_can_1 ?x - gripper ?y - can)
    (gripper_can_2 ?x - gripper ?y - can)
    (freight_surface_0 ?x - freight ?y - surface)
    (freight_surface_1 ?x - freight ?y - surface)
    (freight_gripper_0 ?x - freight ?y - gripper)
    (freight_gripper_1 ?x - freight ?y - gripper)
```

```
    (freight_can_0 ?x - freight ?y - can)
    (freight_can_1 ?x - freight ?y - can)
    (can_surface_0 ?x - can ?y - surface)
    (can_surface_1 ?x - can ?y - surface)
    (aux3_can_surface_1 ?x - can)
    (aux3_freight_can_1 ?x - freight)
    (aux3_freight_surface_1 ?x - freight)
    (aux3_freight_surface_0 ?x - freight)
    (aux3_can_surface_0 ?x - can)
    (aux3_gripper_can_2 ?x - gripper)
    (aux3_freight_gripper_1 ?x - freight)
    (aux3_gripper_can_1 ?x - gripper)
    (aux3_gripper_can_0 ?x - gripper)
    (aux3_freight_gripper_0 ?x - freight)
    (aux3_freight_can_0 ?x - freight)
)

(:action a1
:parameters ( ?can_p1 - can  ?freight_p1 - freight  ?
        surface_extra_p1 - surface  ?gripper_p1 - gripper )
:precondition (and
    (can_surface_1 ?can_p1 ?surface_extra_p1)
    (gripper_can_0 ?gripper_p1 ?can_p1)
    (freight_can_0 ?freight_p1 ?can_p1)
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
```

```
    (aux3_gripper_can_2 ?gripper_p1)
    (aux3_gripper_can_1 ?gripper_p1)
)
:effect (and
    (gripper_can_1 ?gripper_p1 ?can_p1)
    (not (gripper_can_0 ?gripper_p1 ?can_p1))
    (not (gripper_can_2 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_0 ?gripper_p1)
    (not (aux3_gripper_can_1 ?gripper_p1))
 )
)



(:action a2
:parameters ( ?gripper_extra_p1 - gripper  ?can_p1 - can
        ?freight_extra_p1 - freight  ?surface_p1 - surface )
:precondition (and
    (freight_surface_1 ?freight_extra_p1 ?surface_p1)
    (gripper_can_2 ?gripper_extra_p1 ?can_p1)
    (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
    (can_surface_1 ?can_p1 ?surface_p1)
)
:effect (and
    (can_surface_0 ?can_p1 ?surface_p1)
    (not (can_surface_1 ?can_p1 ?surface_p1))
```

```
    (aux3_can_surface_1 ?can_p1)
 )
)

(:action a3
:parameters ( ?gripper_p1 - gripper  ?surface_p1 - surface
        ?freight_p1 - freight )
:precondition (and
    (freight_gripper_1 ?freight_p1 ?gripper_p1)
    (freight_surface_0 ?freight_p1 ?surface_p1)
    (aux3_freight_surface_1 ?freight_p1)
)
:effect (and
    (freight_surface_1 ?freight_p1 ?surface_p1)
    (not (freight_surface_0 ?freight_p1 ?surface_p1))
    (not (aux3_freight_surface_1 ?freight_p1))
 )
)

(:action a4
:parameters ( ?surface_extra_p2 - surface
        ?gripper_p1 - gripper  ?surface_p1 - surface
        ?freight_p1 - freight )
:precondition (and
    (not (= ?surface_extra_p2 ?surface_p1))
    (freight_surface_1 ?freight_p1 ?surface_p1)
```

```
    (freight_surface_0 ?freight_p1 ?surface_extra_p2)
    (freight_gripper_1 ?freight_p1 ?gripper_p1)
)

:effect (and
    (freight_surface_0 ?freight_p1 ?surface_p1)
    (not (freight_surface_1 ?freight_p1 ?surface_p1))
    (aux3_freight_surface_1 ?freight_p1)
 )
)

(:action a5
:parameters ( ?gripper_extra_p1 - gripper  ?can_p1 - can
        ?freight_extra_p1 - freight  ?surface_p1 - surface )
:precondition (and
    (freight_surface_1 ?freight_extra_p1 ?surface_p1)
    (gripper_can_2 ?gripper_extra_p1 ?can_p1)
    (can_surface_0 ?can_p1 ?surface_p1)
    (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
    (aux3_can_surface_1 ?can_p1)
)
:effect (and
    (can_surface_1 ?can_p1 ?surface_p1)
    (not (can_surface_0 ?can_p1 ?surface_p1))
    (not (aux3_can_surface_1 ?can_p1))
 )
```

```
)

(:action a6
:parameters ( ?can_p1 - can  ?freight_p1 - freight
        ?surface_extra_p1 - surface  ?gripper_p1 - gripper )
:precondition (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (freight_can_1 ?freight_p1 ?can_p1)
    (freight_gripper_1 ?freight_p1 ?gripper_p1)
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (aux3_freight_can_0 ?freight_p1)
    (aux3_freight_gripper_0 ?freight_p1)
)
:effect (and
    (freight_can_0 ?freight_p1 ?can_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
    (not (freight_can_1 ?freight_p1 ?can_p1))
    (aux3_freight_can_1 ?freight_p1)
    (aux3_freight_gripper_1 ?freight_p1)
    (not (aux3_freight_can_0 ?freight_p1))
    (not (aux3_freight_gripper_0 ?freight_p1))
 )
)

(:action a7
```

```
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper
        ?surface_extra_p1 - surface  ?freight_p1 - freight )
:precondition (and
    (can_surface_1 ?can_p1 ?surface_extra_p1)
    (freight_can_0 ?freight_p1 ?can_p1)
    (gripper_can_1 ?gripper_p1 ?can_p1)
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (aux3_gripper_can_0 ?gripper_p1)
    (aux3_gripper_can_2 ?gripper_p1)
)
:effect (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (not (gripper_can_0 ?gripper_p1 ?can_p1))
    (not (gripper_can_1 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_1 ?gripper_p1)
    (not (aux3_gripper_can_2 ?gripper_p1))
 )
)

(:action a8
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper
        ?surface_extra_p1 - surface  ?freight_p1 - freight )
:precondition (and
    (freight_can_0 ?freight_p1 ?can_p1)
    (gripper_can_1 ?gripper_p1 ?can_p1)
```

```
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (aux3_gripper_can_0 ?gripper_p1)
    (aux3_gripper_can_2 ?gripper_p1)
)
:effect (and
    (gripper_can_0 ?gripper_p1 ?can_p1)
    (not (gripper_can_2 ?gripper_p1 ?can_p1))
    (not (gripper_can_1 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_1 ?gripper_p1)
    (not (aux3_gripper_can_0 ?gripper_p1))
 )
)

(:action a9
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper
       ?surface_extra_p1 - surface  ?freight_p1 - freight )
:precondition (and
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (freight_can_0 ?freight_p1 ?can_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (aux3_gripper_can_0 ?gripper_p1)
    (aux3_gripper_can_1 ?gripper_p1)
)
:effect (and
```

```
    (gripper_can_1 ?gripper_p1 ?can_p1)
    (not (gripper_can_0 ?gripper_p1 ?can_p1))
    (not (gripper_can_2 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_2 ?gripper_p1)
    (not (aux3_gripper_can_1 ?gripper_p1))
 )
)

(:action a10
:parameters ( ?gripper_p1 - gripper  ?surface_extra_p1 - surface
        ?freight_p1 - freight )
:precondition (and
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (aux3_freight_gripper_1 ?freight_p1)
)
:effect (and
    (freight_gripper_1 ?freight_p1 ?gripper_p1)
    (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
    (aux3_freight_gripper_0 ?freight_p1)
    (not (aux3_freight_gripper_1 ?freight_p1))
 )
)

(:action a11
:parameters ( ?gripper_p1 - gripper  ?surface_extra_p1 - surface
```

```
        ?freight_p1 - freight )
:precondition (and
    (freight_gripper_1 ?freight_p1 ?gripper_p1)
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (aux3_freight_gripper_0 ?freight_p1)
)
:effect (and
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
    (aux3_freight_gripper_1 ?freight_p1)
    (not (aux3_freight_gripper_0 ?freight_p1))
 )
)

(:action a12
:parameters ( ?can_p1 - can  ?freight_p1 - freight
        ?surface_extra_p1 - surface  ?gripper_p1 - gripper )
:precondition (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (can_surface_0 ?can_p1 ?surface_extra_p1)
    (freight_surface_1 ?freight_p1 ?surface_extra_p1)
    (freight_can_0 ?freight_p1 ?can_p1)
    (freight_gripper_0 ?freight_p1 ?gripper_p1)
    (aux3_freight_can_1 ?freight_p1)
    (aux3_freight_gripper_1 ?freight_p1)
)
```

```
:effect (and
    (freight_gripper_1 ?freight_p1 ?gripper_p1)
    (freight_can_1 ?freight_p1 ?can_p1)
    (not (freight_can_0 ?freight_p1 ?can_p1))
    (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
    (aux3_freight_can_0 ?freight_p1)
    (aux3_freight_gripper_0 ?freight_p1)
    (not (aux3_freight_can_1 ?freight_p1))
    (not (aux3_freight_gripper_1 ?freight_p1))
 )
)
)
```

Domain: Packing

```
(define (domain Packing)
(:requirements :strips :typing :equality :conditional-effects
    :existential-preconditions :universal-preconditions)
(:types
    can
    gripper
    surface
)

(:predicates
    (gripper_can_0 ?x - gripper ?y - can)
```

```
    (gripper_can_1 ?x - gripper ?y - can)
    (gripper_can_2 ?x - gripper ?y - can)
    (can_surface_0 ?x - can ?y - surface)
    (can_surface_1 ?x - can ?y - surface)
    (aux3_gripper_can_1 ?x - gripper)
    (aux3_gripper_can_2 ?x - gripper)
    (aux3_gripper_can_0 ?x - gripper)
)

(:action a1
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper )
:precondition (and
    (gripper_can_0 ?gripper_p1 ?can_p1)
    (aux3_gripper_can_2 ?gripper_p1)
    (aux3_gripper_can_1 ?gripper_p1)
)
:effect (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (not (gripper_can_0 ?gripper_p1 ?can_p1))
    (not (gripper_can_1 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_0 ?gripper_p1)
    (not (aux3_gripper_can_2 ?gripper_p1))
 )
)

(:action a2
```

```
:parameters ( ?can_p1 - can  ?surface_extra_p1 - surface
        ?gripper_p1 - gripper )
:precondition (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (can_surface_1 ?can_p1 ?surface_extra_p1)
    (aux3_gripper_can_0 ?gripper_p1)
    (aux3_gripper_can_1 ?gripper_p1)
)
:effect (and
    (gripper_can_0 ?gripper_p1 ?can_p1)
    (not (gripper_can_2 ?gripper_p1 ?can_p1))
    (not (gripper_can_1 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_2 ?gripper_p1)
    (not (aux3_gripper_can_0 ?gripper_p1))
 )
)

(:action a3
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper )
:precondition (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (aux3_gripper_can_0 ?gripper_p1)
    (aux3_gripper_can_1 ?gripper_p1)
)
:effect (and
    (gripper_can_1 ?gripper_p1 ?can_p1)
```

```
    (not (gripper_can_0 ?gripper_p1 ?can_p1))
    (not (gripper_can_2 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_2 ?gripper_p1)
    (not (aux3_gripper_can_1 ?gripper_p1))
 )
)

(:action a4
:parameters ( ?can_p1 - can  ?surface_extra_p1 - surface
    ?gripper_p1 - gripper )
:precondition (and
    (can_surface_1 ?can_p1 ?surface_extra_p1)
    (gripper_can_1 ?gripper_p1 ?can_p1)
    (aux3_gripper_can_0 ?gripper_p1)
    (aux3_gripper_can_2 ?gripper_p1)
)
:effect (and
    (gripper_can_2 ?gripper_p1 ?can_p1)
    (not (gripper_can_0 ?gripper_p1 ?can_p1))
    (not (gripper_can_1 ?gripper_p1 ?can_p1))
    (aux3_gripper_can_1 ?gripper_p1)
    (not (aux3_gripper_can_2 ?gripper_p1))
 )
)

(:action a5
```

```
:parameters ( ?can_p1 - can  ?gripper_extra_p1 - gripper
    ?surface_p1 - surface )
:precondition (and
    (can_surface_0 ?can_p1 ?surface_p1)
    (gripper_can_1 ?gripper_extra_p1 ?can_p1)
)
:effect (and
    (can_surface_1 ?can_p1 ?surface_p1)
    (not (can_surface_0 ?can_p1 ?surface_p1))
 )
)
)
```

Chapter 7

OTHER APPLICATIONS

This chapter presents JEDAI – an application of approaches developed as part of this thesis. JEDAI uses various concepts of planning and robotics to educate non-experts about planning in AI and robotics. It also allows non-experts to interact with robot systems in a simulator and enables them to learn about planning in robotics.

7.1 Motivation

AI systems are increasingly common in everyday life, where they can be used by laypersons who may not understand how these autonomous systems work or what they can and cannot do. This problem is particularly salient in cases of taskable AI systems whose functionality can change based on the tasks they are performing. In this work, we present an AI system JEDAI (JEDAI Explains Decision-Making AI) that can be used in outreach and educational efforts to help laypersons learn how to provide AI systems with new tasks, debug such systems, and understand their capabilities.

The research ideas brought together in JEDAI address three key technical challenges: (i) abstracting a robot's functionalities into high-level actions (capabilities) that the user can more easily understand; (ii) converting the user-understandable capabilities into low-level motion plans that a robot can execute; and (iii) explaining errors in a manner sensitive to the user's current level of knowledge so as to make the robot's capabilities and limitations clear.

JEDAI utilizes recent work in explainable AI and integrated task and motion planning to address these challenges and provides a simple interface to support accessibility. Users select a domain and an associated task, after which they create a plan consisting of high-level

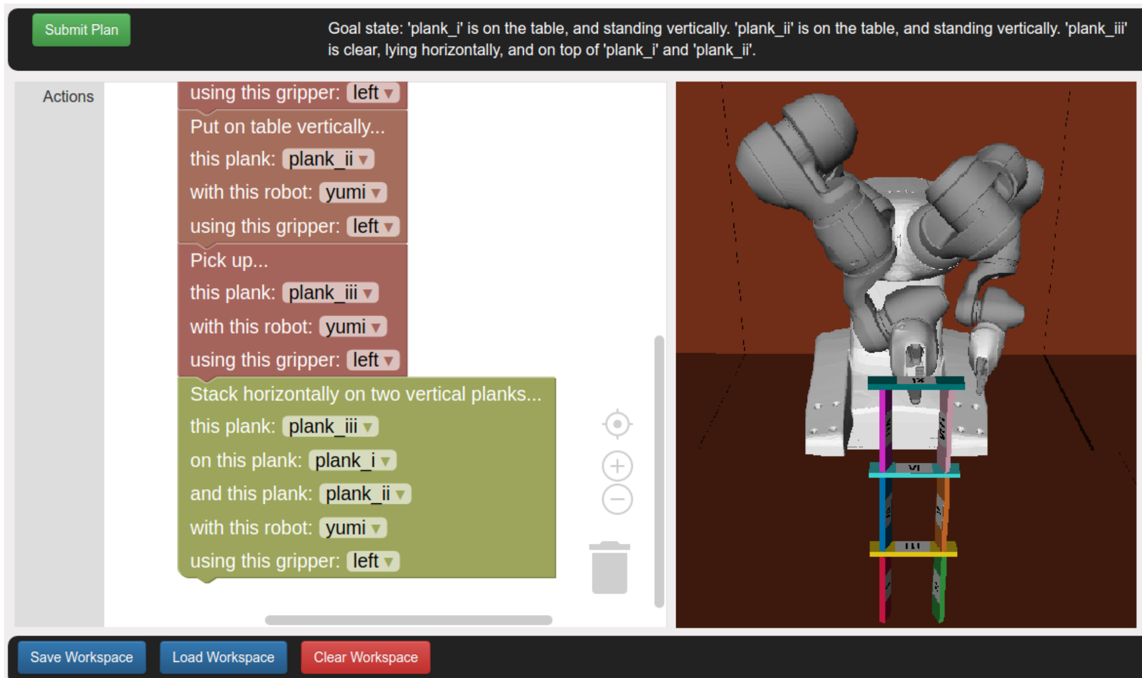


Figure 7.1: JEDAI system with a Blockly-based plan creator on the left and a simulator window on the right.

actions (Fig. 7.1 left) to complete the task. The user puts together a plan in a drag-and-drop workspace, built with the Blockly visual programming library Google (2017). JEDAI validates this plan using the Hierarchical Expertise Level Modeling algorithm (HELM) Sreedharan *et al.* (2018, 2021). If the plan contains any errors, HELM computes a user-specific explanation of why the plan would fail. JEDAI converts such explanations to natural language, thus helping to identify and fix any gaps in the user's understanding. Whereas, if the plan given by the user is a correct solution to the current task, JEDAI uses a task and motion planner ATM-MDP Shah *et al.* (2020); Shah and Srivastava (2021) to convert the high-level plan, that the user understands, to a low-level motion plan that the robot can execute. The user is shown the execution of this low-level motion plan by the robot in a simulated environment (Fig. 7.1 right).

Prior work on the topic includes approaches that solve the three technical challenges mentioned earlier *in isolation*. This includes tools for: providing visualizations or ani-

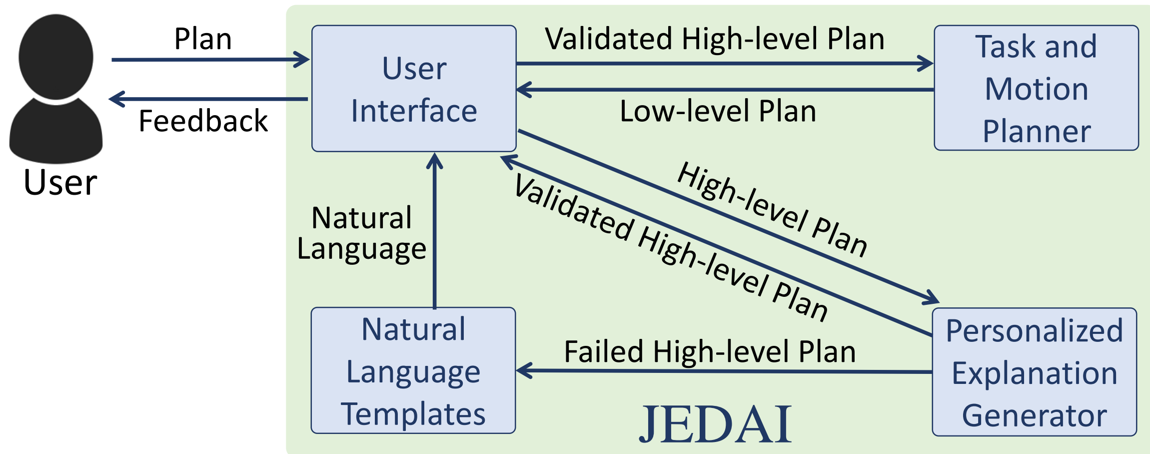


Figure 7.2: Architecture of JEDAI showing interaction between the four core components.

mations of standard planning domains Magnaguagno *et al.* (2017); Chen *et al.* (2019); Aguinaldo and Regli (2021); Dvorak *et al.* (2021); De Pellegrin and Petrick (2021); Roberts *et al.* (2021); making it easier for non-expert users to program robots with low-level actions Krishnamoorthy and Kapila (2016); Weintrop *et al.* (2018); Huang *et al.* (2020a); Winterer *et al.* (2020); and generating explanations for plans provided by the users Grover *et al.* (2020); Karthik *et al.* (2022); Brandao *et al.* (2021); Kumar *et al.* (2022). In addition, none of these works make the instructions easier for the user, have the ability to automatically compute user-aligned explanations, and work with real robots (or their simulators) at the same time. JEDAI addresses all three challenges in tandem by using 3D simulations for domains with real robots and their actual constraints and providing personalized explanations that inform a user of any mistake they make while using the system.

7.2 Architecture

Fig. 7.2 shows the four core components of the JEDAI framework: (i) user interface, (ii) task and motion planner, (iii) personalized explanation generator, and (iv) natural language templates. We now describe each component in detail.

User interface JEDAI's UI (Fig. 7.1) is made to be unintimidating and easy to use. The Blockly visual programming interface is used to facilitate this. JEDAI generates a separate interconnecting block for each high-level action, and action parameters are picked from drop-down selection fields that display type-consistent options for each parameter. Users can drag-and-drop these actions and select different arguments to create a high-level plan.

Personalized explanation generator Users will sometimes make mistakes when planning, either failing to achieve goal conditions or applying actions before the necessary preconditions are satisfied. For inexperienced users in particular, these mistakes may stem from an incomplete understanding of the task's requirements or the robot's capabilities. JEDAI assists users in apprehending these details by providing explanations personalized to each user.

Explanations in the context of this work are of two types: (i) non-achieved goal conditions, and (ii) violation of a precondition of an action. JEDAI validates the plan submitted by the user to check if it achieves all goal conditions. If it fails to achieve any goal condition, the user is informed about it. JEDAI uses HELM to compute user-specific contrastive explanations in order to explain any unmet precondition in an action used in the user's plan. HELM does this by using the plan submitted by the user to estimate the user's understanding of the robot's model and then uses the estimated model to compute the personalized explanations. In case of multiple errors in the user's plan, HELM generates explanation for one of the errors. This is because explaining the reason for more than one errors might be unnecessary and in the worst case might leave the user feeling overwhelmed Miller (2019). An error is selected for explanation by HELM based on optimizing a cost function that indicates the relative difficulty of concept understandability which can be changed to reflect different users' background knowledge.

Natural language templates Even with a user-friendly interface and personalized explanations for errors in abstract plans, domain model syntax used for interaction with ATM-MDP presents a significant barrier to a non-expert trying to understand the state of an environment and the capabilities of a robot. To alleviate this, JEDAI uses language templates that use the structure of the planning formalism for generating natural language descriptions for goals, actions, and explanations. E.g., the action *"pickup (plank_i gripper_left)"* can be described in natural language as "pick up *plank_i* with *the left gripper*". Currently, we use hand-written templates for these translations, but an automated approach can also be used.

Task and motion planner JEDAI uses ATM-MDP to convert the high-level plan submitted by the user into sequences of low-level primitive actions that a robot can execute.

ATM-MDP uses sampling-based motion planners to provide a probabilistically complete approach to hierarchical planning. High-level plans are refined by computing feasible motion plans for each high-level action. If an action does not accept any valid refinement due to discrepancies between the symbolic state and the low-level environment, it reports the failure back to JEDAI. If all actions in the high-level plan are refined successfully, the plan's execution is shown using the OpenRAVE simulator Diankov (2010).

Implementation Any custom domain can be set up with JEDAI. We provide five built-in domains, each with one of YuMi ABB (2015) or Fetch Wise *et al.* (2016) robots. Each domain contains a set of problems that the users can attempt to solve and low-level environments corresponding to these problems. Source code for the framework, an already setup virtual machine, and the documentation are available at: `https://github.com/aair-lab/AAIR-JEDAI`. A video demonstrating JEDAI's working is available at: `https://youtu.be/MQdoikcnhbY`.

7.3 Conclusion

We demonstrated a novel AI tool JEDAI for helping people understand the capabilities of an arbitrary AI system and enabling them to work with such systems. JEDAI converts the user's input plans to low level motion plans executable by the robot if it is correct, or explains to the user any error in the plan if it is incorrect. JEDAI works with off-the-shelf task and motion planners and explanation generators. This structure allows it to scale automatically with improvements in either of these active research areas. JEDAI's vizualization-based interface could also be used to foster trust in AI systems Beauxis-Aussalet *et al.* (2021).

JEDAI uses predefined abstractions to verify plans provided by the user. In the future, we plan on extending it to learn abstractions automatically (Shah and Srivastava, 2022b). JEDAI could also be extended as an interface for assessing an agent's functionalities and capabilities by interrogating the agent (Verma *et al.*, 2021; Nayyar *et al.*, 2022; Verma *et al.*, 2022) as well as to work as an interface that makes AI systems compliant with Level II assistive AI – systems that makes it easy for operators to learn how to use them safely Srivastava (2021). Extending this tool for working in non-stationary settings, and generating natural language descriptions of predicates and actions autonomously are a few other promising directions of future work.

Chapter 8

CONCLUSION AND FUTURE WORK

This thesis has addressed the crucial challenge of automating the creation of world models for robot planning, a task traditionally reliant on human expertise and intuition. By leveraging unannotated and unsegmented low-level trajectories, our approaches have successfully learned symbolic and interpretable world models, which are not only effective but also generalizable across diverse environments and problems.

We have introduced innovative methodologies across various chapters, each contributing significantly to the advancement of robot planning:

Stochastic task and motion planning (Chapter 3) Our novel anytime approach has revolutionized policies for stochastic task and motion planning problems, effectively handling the inherent noise in robot actions and ensuring efficient execution even before a complete solution is computed.

Automatically learning zero-shot abstractions (Chapter 4) By automating the inventing state and action abstractions for deterministic motion planning, we have eliminated the need for human-provided world models. Our approach, coupled with a hierarchical multi-source multi-directional planner, has demonstrated remarkable performance improvements over existing methods.

Zero-shot option invention for stochastic motion planning (Chapter 5) Addressing the imperfections in real-world robot actuation, our methodology for inventing options in a zero-shot manner has significantly enhanced the efficacy of stochastic motion planning. The integration of symbolic planning and deep reinforcement learning has yielded promis-

ing theoretical guarantees and empirical results.

Automatically inventing relational world models (Chapter 6) Our approach to automatically inventing relational abstractions has paved the way for interpretable and efficient hierarchical robot planning. By utilizing minimal input data in the form of unannotated and unsegmented robot trajectories, we have demonstrated the practicality and effectiveness of our methodology across various scenarios.

Furthermore, we have presented an application, JEDAI (Chapter 7), aimed at educating students and non-experts about AI planning and robotics, showcasing the real-world applicability and broader impact of our research.

In essence, this thesis marks a significant step towards autonomous and adaptable robot planning systems, with contributions spanning from theoretical advancements to practical implementations. The developed methodologies offer promising avenues for future research and applications in the field of robotics and artificial intelligence.

Future Work Now, we highlight a few future directions that we aim to explore.

Probabilistic world models Our approach for learning relational world models currently assumes a deterministic setting, i.e., learned actions only have a single possible outcome. However, as mentioned earlier, in most real-world scenarios, the robot's actions are stochastic. Therefore, as part of the future work, we aim to learn world models for stochastic settings.

Stronger guarantees on learned world models Currently, our approach relies on associative learning with passively collected data for learning action models, which are prone to making errors. This can lead to inaccurate models and a lack of strong theoretical guarantees on the learned models. We aim to use active learning to learn accurate models and

provide guarantees on learned action models.

Learning in the observation space and partial observability Our approach assumes full observability. This means that the poses of each object in the environment and the configurations of each robot are precisely known while learning abstractions and using them in the real world. However, in practice, accurate poses of the objects are difficult to obtain. Therefore, we aim to develop approaches that do not rely on perfect perception and handle partial and imperfect observability in high-dimensional observation space.

REFERENCES

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning", in "Proc. OSDI, 2016", (2016).

ABB, "ABB YuMi - IRB 14000", `https://new.abb.com/products/robotics/collaborative-robots/irb-14000-yumi` (2015).

Abdelhadi, L. and D. Cherki, "A new transformed stochastic shortest path with dead ends and energy constraint", International Journal of Advanced Science and Technology **129**, 43–58 (2019).

Aguinaldo, A. and W. Regli, "A Graphical Model-Based Representation for Classical AI Plans using Category Theory", in "ICAPS 2021 Workshop on Explainable AI Planning", (2021).

Ahmetoglu, A., M. Y. Seker, J. Piater, E. Oztop and E. Ugur, "DeepSym: Deep symbol generation and rule learning for planning from unsupervised robot interaction", JAIR **75**, 709–745 (2022).

Ahn, M., A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan and A. Zeng, "Do as I can, not as I say: Grounding language in robotic affordances", in "Proc. CoRL", (2023).

Aineto, D., S. J. Celorrio and E. Onaindia, "Learning action models with minimal observability", AIJ **275**, 104–137 (2019).

Alterovitz, R., T. Siméon and K. Goldberg, "The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty", in "Robotics: Science and systems", (2007).

Amos, B., I. Jimenez, J. Sacks, B. Boots and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control", in "Proc. NeurIPS", (2018).

Andersen, G. and G. Konidaris, "Active exploration for learning symbolic representations", in "Proc. NeurIPS", (2017).

Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel and W. Zaremba, "Hindsight experience replay", in "Proc. NeurIPS", (2017).

Asai, M., H. Kajino, A. Fukunaga and C. Muise, "Classical planning in deep latent space", JAIR **74**, 1599–1686 (2022).

Bacchus, F. and F. Kabanza, "Using temporal logics to express search control knowledge for planning", Artificial intelligence **116**, 1-2, 123–191 (2000).

Bacon, P.-L., J. Harb and D. Precup, "The option-critic architecture", in "Proc. AAAI", (2017).

Bagaria, A. and G. Konidaris, "Option discovery using deep skill chaining", in "Proc. ICLR", (2020).

Bagaria, A., J. K. Senthil and G. Konidaris, "Skill discovery for exploration and planning using deep skill graphs", in "Proc. ICML", (2021).

Bai, A., S. Srivastava and S. J. Russell, "Markovian state and action abstractions for MDPs via hierarchical MCTS.", in "Proc. International Joint Conference on Artificial Intelligence", (2016).

Barto, A., S. Bradtke and S. Singh, "Learning to act using real-time dynamic programming", Artificial Intelligence **72**, 81–138 (1993).

Beauxis-Aussalet, E., M. Behrisch, R. Borgo, D. H. Chau, C. Collins, D. Ebert, M. El-Assady, A. Endert, D. A. Keim, J. Kohlhammer, D. Oelke, J. Peltonen, M. Riveiro, T. Schreck, H. Strobelt and J. J. van Wijk, "The role of interactive visualization in fostering trust in ai", IEEE Computer Graphics and Applications **41**, 6, 7–12 (2021).

Bercher, P., S. Keen and S. Biundo, "Hybrid planning heuristics based on task decomposition graphs", in "Seventh Annual Symposium on Combinatorial Search", (2014).

Berenson, D., S. S. Srinivasa, D. Ferguson and J. J. Kuffner, "Manipulation planning on constraint manifolds", in "Proc. International Conference on Robotics and Automation", (2009).

Berg, J. v. d., S. Patil and R. Alterovitz, "Motion planning under uncertainty using differential dynamic programming in belief space", in "Robotics Research", pp. 473–490 (Springer, 2017).

Birgui Sekou, T., M. Hidane, J. Olivier and H. Cardot, "Retinal blood vessel segmentation using a fully convolutional network – transfer learning from patch- to image-level", in "Proc. MLMI, 2018", (2018).

Blum, A. L. and M. L. Furst, "Fast planning through planning graph analysis", Artificial intelligence **90**, 1-2, 281–300 (1997).

Bonet, B. and H. Geffner, "Planning as heuristic search", Artificial Intelligence **129**, 1-2, 5–33 (2001).

Bonet, B. and H. Geffner, "Labeled rtdp: Improving the convergence of real-time dynamic programming.", in "Proc. International Conference on Automated Planning and Scheduling", (2003).

Bonet, B. and H. Geffner, "Learning first-order symbolic representations for planning from the structure of the state space", in "Proc. ECAI", (2019).

Bonet, B., H. Palacios and H. Geffner, "Automatic derivation of memoryless policies and finite-state controllers using classical planners", in "Proc. International Conference on Automated Planning and Scheduling", (2009).

Brandao, M., G. Canal, S. Krivić and D. Magazzeni, "Towards Providing Explanations for Robot Motion Planning", in "Proc. ICRA", (2021).

Brock, O. and L. E. Kavraki, "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces", in "Proc. ICRA, 2001", (2001).

Brohan, A., N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. Ryoo, G. Salazar, P. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu and B. Zitkovich, "RT-1: Robotics transformer for real-world control at scale", arXiv:2212.06817 (2022).

Brunskill, E. and L. Li, "Pac-inspired option discovery in lifelong reinforcement learning", in "Proc. ICML", (2014).

Bryce, D., J. Benton and M. W. Boldt, "Maintaining evolving domain models", in "Proc. IJCAI", (2016).

Cambon, S., R. Alami and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning", IJRR **28**, 104–126 (2009).

Čertický, M., "Real-Time Action Model Learning with Online Algorithm 3SG", Applied AI **28**, 7, 690–711 (2014).

Chamzas, C., Z. Kingston, C. Quintero-Peña, A. Shrivastava and L. E. Kavraki, "Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions", (2020).

Chazelle, B., "Approximation and decomposition of shapes", Algorithmic and Geometric Aspects of Robotics **1**, 145–185 (1985).

Chen, G., Y. Ding, H. Edwards, C. H. Chau, S. Hou, G. Johnson, M. Sharukh Syed, H. Tang, Y. Wu, Y. Yan, T. Gil and L. Nir, "Planimation", in "ICAPS 2019 System Demonstrations", (2019).

Chen, X., Y. Duan, R. Houthooft, J. Schulman, I. Sutskever and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets", in "Proc. NeurIPS, 2016", (2016).

Cheng, S., C. Garrett, A. Mandlekar and D. Xu, "NOD-TAMP: Multi-step manipulation planning with neural object descriptors", in "CoRL 2023 LEAP Workshop", (2023).

Chitnis, R., T. Silver, J. B. Tenenbaum, T. Lozano-Pérez and L. P. Kaelbling, "Learning neuro-symbolic relational transition models for bilevel planning", in "Proc. IROS", (2022).

Cresswell, S., T. McCluskey and M. West, "Acquisition of object-centred domain models from planning examples", in "Proc. ICAPS", (2009).

Şimşek, O., A. P. Wolfe and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning", in "Proc. ICML, 2005", (2005).

Czechowski, K., T. Odrzygóźdź, M. Zbysiński, M. Zawalski, K. Olejnik, Y. Wu, Ł. Kuciński and P. Miłoś, "Subgoal search for complex reasoning tasks", in "Proc. NeurIPS", (2021).

Dantam, N., Z. Kingston and S. Chaudhuriand L. Kavraki, "An incremental constraint-based framework for task and motion planning", IJRR **37**, 10, 1134–1151 (2018).

De Pellegrin, E. and R. P. A. Petrick, "PDSim: Simulating Classical Planning Domains with the Unity Game Engine", in "ICAPS 2021 System Demonstrations", (2021).

Denny, J. and N. M. Amato, "Toggle prm: A coordinated mapping of c-free and c-obstacle in arbitrary dimension", vol. 86, pp. 297–312 (Springer, Berlin, Heidelberg., 2013).

Devin, C., A. Gupta, T. Darrell, P. Abbeel and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer", in "Proceedings of International Conference on Robotics and Automation", (2017).

Diankov, R., *Automated Construction of Robotic Manipulation Programs*, Ph.D. thesis, Carnegie Mellon University (2010).

Dijkstra, E. W., "A note on two problems in connexion with graphs", Numerische mathematik **1**, 1, 269–271 (1959).

Ding, Y., X. Zhang, C. Paxton and S. Zhang, "Task and motion planning with large language models for object rearrangement", in "Proc. IROS", (2023).

Du, Y., D. Hsu, H. Kurniawati, W. Sun, L. Sylvie, C. Ong and S. W. Png, "A pomdp approach to robot motion planning under uncertainty", in "In International Conference on Automated Planning & Scheduling, Workshop on Solving Real-World POMDP Problems", (Citeseer, 2010).

Dvorak, F., A. Agarwal and N. Baklanov, "Visual Planning Domain Design for PDDL using Blockly", in "ICAPS 2021 System Demonstrations", (2021).

Ebert, F., C. Finn, S. Dasari, A. Xie, A. Lee and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control", arXiv preprint arXiv:1812.00568 (2018).

Erol, K., J. A. Hendler and D. S. Nau, "Semantics for hierarchical task-network planning", Tech. rep., MARYLAND UNIV COLLEGE PARK INST FOR SYSTEMS RESEARCH (1995).

Eysenbach, B., R. R. Salakhutdinov and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning", in "Proc. NeurIPS", (2019).

Fang, X., C. R. Garrett, C. Eppner, T. Lozano-Pérez, L. P. Kaelbling and D. Fox, "DiMSam: Diffusion models as samplers for task and motion planning under partial observability", in "CoRL 2023 LEAP Workshop", (2023).

Finn, C., X. Y. Tan, Y. Duan, T. Darrell, S. Levine and P. Abbeel, "Deep spatial autoencoders for visuomotor learning", in "Proc. ICRA", (2016).

Fox, M. and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains", Jornal of Artificial Intelligence Research **20**, 61–124 (2003).

Gal, Y., R. McAllister and C. E. Rasmussen, "Improving PILCO with bayesian neural network dynamics models", in "Data-efficient machine learning workshop, ICML", (2016).

Garrett, C., T. Lozano-Pérez and L. Kaelbling, "FFRob: An efficient heuristic for task and motion planning", in "Proc. WAFR", (2015).

Garrett, C., T. Lozano-Pérez and L. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning", in "Proc. ICAPS", (2020).

Garrett, C. R., R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning", Annual Review of Control, Robotics, and Autonomous Systems **4**, 1, null (2021).

Garrett, C. R., C. Paxton, T. Lozano-Pérez, L. P. Kaelbling and D. Fox, "Online replanning in belief space for partially observable task and motion problems", in "2020 IEEE International Conference on Robotics and Automation (ICRA)", pp. 5678–5684 (IEEE, 2020).

Google, "Blockly", `https://github.com/google/blockly` (2017).

Gopalan, N., M. Littman, J. MacGlashan, S. Squire, S. Tellex, J. Winder, L. Wong *et al.*, "Planning with abstract markov decision processes", in "Proceedings of the International Conference on Automated Planning and Scheduling", (2017).

Goyal, A., J. Xu, Y. Guo, V. Blukis, Y.-W. Chao and D. Fox, "RVT: Robotic view transformer for 3D object manipulation", in "Proc. CoRL", (2023).

Grover, S., S. Sengupta, T. Chakraborti, A. P. Mishra and S. Kambhampati, "RADAR: Automated Task Planning for Proactive Decision Support", Human–Computer Interaction **35**, 5-6, 387–412 (2020).

Haarnoja, T., A. Zhou, P. Abbeel and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", in "Proc. ICML", (2018).

Hadfield-Menell, D., E. Groshev, R. Chitnis and P. Abbeel, "Modular task and motion planning in belief space", in "Proc. International Conference on Intelligent Robots and Systems", (2015).

Hafner, D., T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee and J. Davidson, “Learning latent dynamics for planning from pixels”, in “Proc. ICML”, (2019).

Hafner, D., J. Pasukonis, J. Ba and T. Lillicrap, “Mastering diverse domains through world models”, arXiv preprint arXiv:2301.04104 (2023).

Hansen, E. A. and S. Zilberstein, “LAO*: A heuristic search algorithm that finds solutions with loops”, Artificial Intelligence **129**, 1-2, 35–62 (2001).

Hausknecht, M. and P. Stone, “Deep reinforcement learning in parameterized action space”, in “Proc. International Conference on Learning Representations”, (2016).

Helmert, M., “The fast downward planning system”, JAIR **26**, 191–246 (2006).

Henaff, M., W. F. Whitney and Y. LeCun, “Model-based planning with discrete and continuous actions”, arXiv preprint arXiv:1705.07177 (2017).

Hertle, A., C. Dornhege, T. Keller and B. Nebel, “Planning with semantic attachments: An object-oriented view”, in “Proc. ECAI”, (2012).

Hibbard, M., A. P. Vinod, J. Quattrociocchi and U. Topcu, “Safely: Safe stochastic motion planning under constrained sensing via duality”, arXiv preprint arXiv:2203.02816 (2022).

Hoffmann, J., “Ff: The fast-forward planning system”, AI magazine **22**, 3, 57–57 (2001).

Holte, R. C., M. B. Perez, R. M. Zimmer and A. J. MacDonald, “Hierarchical A*: Searching abstraction hierarchies efficiently”, in “Proc. AAAI”, pp. 530–535 (1996).

Hostetler, J., A. Fern and T. Dietterich, “State aggregation in monte carlo tree search.”, in “Proc. Association for the Advancement of Artificial Intelligence”, (2014).

Hou, L., D. Samaras, T. M. Kurc, Y. Gao, J. E. Davis and J. H. Saltz, “Patch-based convolutional neural network for whole slide tissue image classification”, in “Proc. CVPR, 2016”, (2016).

Hu, Y. and G. De Giacomo, “Generalized planning: Synthesizing plans that work for multiple environments”, in “Proc. International Joint Conference on Artificial Intelligence”, (2011).

Huang, G., P. S. Rao, M.-H. Wu, X. Qian, S. Y. Nof, K. Ramani and A. J. Quinn, “Vipo: Spatial-Visual Programming with Functions for Robot-IoT Workflows”, in “Proc. CHI”, (2020a).

Huang, W., P. Abbeel, D. Pathak and I. Mordatch, “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents”, in “Proc. ICML”, (2022).

Huang, W., I. Mordatch and D. Pathak, “One policy to control them all: Shared modular policies for agent-agnostic control”, in “Proceedings of International Conference on Machine Learning”, (2020b).

Huang, W., C. Wang, R. Zhang, Y. Li, J. Wu and L. Fei-Fei, "VoxPoser: Composable 3D value maps for robotic manipulation with language models", in "Proc. CoRL", (2023a).

Huang, W., F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models", in "Proc. CoRL", (2023b).

Huynh, V. A., S. Karaman and E. Frazzoli, "An incremental sampling-based algorithm for stochastic optimal control", The International Journal of Robotics Research **35**, 4, 305–333 (2016).

Ichter, B., J. Harrison and M. Pavone, "Learning sampling distributions for robot motion planning", in "Proc. ICRA, 2018", (2018).

Ichter, B., E. Schmerling, T.-W. E. Lee and A. Faust, "Learned critical probabilistic roadmaps for robotic motion planning", in "Proc. ICRA, 2020", (2020).

James, S., B. Rosman and G. Konidaris, "Learning portable representations for high-level planning", in "Proc. ICML", (2020).

Jinnai, Y., D. Abel, D. Hershkowitz, M. Littman and G. Konidaris, "Finding options that minimize planning time", in "Proc. ICML", (2019).

Jurgenson, T., O. Avner, E. Groshev and A. Tamar, "Sub-goal trees a framework for goal-based reinforcement learning", in "International Conference on Machine Learning", pp. 5020–5030 (PMLR, 2020).

Jurgenson, T. and A. Tamar, "Harnessing reinforcement learning for neural motion planning", in "Proc. RSS", (2019).

Kaelbling, L. P. and T. Lozano-Pérez, "Hierarchical task and motion planning in the now", in "Proc. International Conference on Robotics and Automation", (2011).

Kambhampati, S., A. Mali and B. Srivastava, "Hybrid planning for partially hierarchical domains", in "AAAI/IAAI", (1998).

Karia, R., R. K. Nayyar and S. Srivastava, "Learning generalized policy automata for relational stochastic shortest path problems", Advances in Neural Information Processing Systems (2022).

Karia, R. and S. Srivastava, "Learning generalized relational heuristic metworks for model-agnostic planning", in "Proc. AAAI, 2021", (2021).

Karthik, V., S. Sreedharan, S. Sengupta and S. Kambhampati, "RADAR-X: An Interactive Mixed Initiative Planning Interface Pairing Contrastive Explanations and Revised Plan Suggestions", in "Proc. ICAPS", (2022).

Kavraki, L. E., P. Svestka, J.-C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE transactions on Robotics and Automation **12**, 4, 566–580 (1996).

Kim, J., Y. Seo and J. Shin, “Landmark-guided subgoal generation in hierarchical reinforcement learning”, in “Proc. NeruIPS”, (2021).

Kingma, D. and J. Ba, “Adam: A method for stochastic optimization”, Proc. ICLR, 2014 (2014).

Knoblock, C. A., “Learning abstraction hierarchies for problem solving.”, in “AAAI”, pp. 923–928 (1990).

Kokel, H., A. Manoharan, S. Natarajan, B. Ravindran and P. Tadepalli, “Reprel: Integrating relational planning and reinforcement learning for effective abstraction”, in “Proc. ICAPS”, (2021).

Konidaris, G., L. P. Kaelbling and T. Lozano-Pérez, “Constructing symbolic representations for high-level planning”, in “Proc. AAAI”, (2014).

Konidaris, G., L. P. Kaelbling and T. Lozano-Pérez, “Symbol acquisition for probabilistic high-level planning”, in “Proc. IJCAI”, (2015).

Konidaris, G., L. P. Kaelbling and T. Lozano-Perez, “From skills to symbols: Learning symbolic representations for abstract high-level planning”, Journal of Artificial Intelligence Research **61**, 215–289 (2018).

Koval, M., “Prpy”, `https://github.com/personalrobotics/prpy` (2015).

Krishnamoorthy, S. P. and V. Kapila, “Using A Visual Programming Environment and Custom Robots to Learn C Programming and K-12 STEM Concepts”, in “Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education”, (2016).

Kuffner, J. J. and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning”, in “Proc. International Conference on Robotics and Automation”, (2000).

Kuffner, J. J. and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning”, in “Proc. ICRA, 2000.”, (2000).

Kulkarni, T. D., K. Narasimhan, A. Saeedi and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation”, in “Proc. NeurIPS”, (2016).

Kumar, A., S. L. Vasileiou, M. Bancilhon, A. Ottley and W. Yeoh, “VizXP: A Visualization Framework for Conveying Explanations to Users in Model Reconciliation Problems”, in “Proc. ICAPS”, (2022).

Kumar, N., W. McClinton, R. Chitnis, T. Silver, T. Lozano-Pérez and L. P. Kaelbling, “Learning efficient abstract planning models that choose what to predict”, in “Proc. CoRL”, (2023).

Kumar, R., A. Mandalika, S. Choudhury and S. Srinivasa, “Lego: Leveraging experience in roadmap generation for sampling-based planning”, in “Proc. IROS”, (2019).

Kurniawati, H., T. Bandyopadhyay and N. M. Patrikalakis, "Global motion planning under uncertain motion, sensing, and environment map", Autonomous Robots **33**, 3, 255–272 (2012).

Kurutach, T., A. Tamar, G. Yang, S. Russell and P. Abbeel, "Learning plannable representations with causal InfoGAN", (2018).

Kwon, M., S. M. Xie, K. Bullard and D. Sadigh, "Reward design with language models", in "Proc. ICLR", (2023).

Lagriffoul, F., N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava and L. E. Kavraki, "Platform-independent benchmarks for task and motion planning", IEEE Robotics and Automation Letters **3**, 4, 3765–3772 (2018).

Lamanna, L., A. Saetti, L. Serafini, A. Gerevini and P. Traverso, "Online Learning of Action Models for PDDL Planning", in "Proc. IJCAI", (2021).

Lavalle, S. M., "Rapidly-exploring random trees: A new tool for path planning", Tech. rep. (1998).

LaValle, S. M., "Rapidly-exploring random trees: A new tool for path planning", (1998).

LaValle, S. M., *Planning Algorithms* (Cambridge University Press, USA, 2006).

Levine, S., C. Finn, T. Darrell and P. Abbeel, "End-to-end training of deep visuomotor policies", The Journal of Machine Learning Research **17**, 1, 1334–1373 (2016).

Levy, A., G. Konidaris, R. Platt and K. Saenko, "Learning multi-level hierarchies with hindsight", in "Proc. ICLR", (2019).

Li, L., T. J. Walsh and M. L. Littman, "Towards a unified theory of state abstraction for mdps.", in "Proc. International Symposium on Artificial Intelligence and Mathematics", (2006).

Liang, J. and A. Boularias, "Learning category-level manipulation tasks from point clouds with dynamic graph cnns", in "Proc. ICRA", (2023).

Liang, J., W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence and A. Zeng, "Code as policies: Language model programs for embodied control", in "Proc. ICRA", (2023).

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning", in "Proc. ICLR", (2016).

Lin, K., C. Agia, T. Migimatsu, M. Pavone and J. Bohg, "Text2Motion: From natural language instructions to feasible plans", Autonomous Robots (2023).

Liu, B., Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas and P. Stone, "LLM+P: Empowering large language models with optimal planning proficiency", arXiv:2304.11477 (2023a).

Liu, K., M. Stadler and N. Roy, "Learned sampling distributions for efficient planning in hybrid geometric and object-level representations", in "Proc. ICRA, 2020", (2020).

Liu, W., Y. Du, T. Hermans, S. Chernova and C. Paxton, "Structdiffusion: Language-guided creation of physically-valid structures using unseen objects", in "Proc. RSS", (2023b).

Lowerre, B. T., *The Harpy Speech Recognition System.* (Carnegie Mellon University, 1976).

Lu, X., Z. Lin, X. Shen, R. Mech and J. Z. Wang, "Deep multi-patch aggregation network for image style, aesthetics, and quality estimation", in "Proc. ICCV, 2015", (2015).

Lyu, D., F. Yang, B. Liu and S. Gustafson, "Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning", in "Proc. AAAI", (2019).

Magnaguagno, M. C., R. Fraga Pereira, M. D. Móre and F. R. Meneguzzi, "WEB PLANNER: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search", in "ICAPS 2017 Workshop on User Interfaces and Scheduling and Planning", (2017).

Marecki, J., Z. Topol, M. Tambe *et al.*, "A fast analytical algorithm for mdps with continuous state sp", in "Proc. Autonomous Agents and Multiagent Systems", (2006).

Marthi, B., S. Russell and J. A. Wolfe, "Angelic semantics for high-level actions.", in "ICAPS", pp. 232–239 (2007a).

Marthi, B., S. J. Russell and J. A. Wolfe, "Angelic semantics for high-level actions.", in "Proc. International Conference on Automated Planning and Scheduling", (2007b).

Martín-Martín, R., M. A. Lee, R. Gardner, S. Savarese, J. Bohg and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks", in "Proceedings of International Conference on Intelligent Robots and Systems (IROS)", (2019).

McDermott, D., M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. S. Weld and D. Wilkins, "PDDL – The Planning Domain Definition Language", Tech. Rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998).

Miller, T., "Explanation in Artificial Intelligence: Insights from the Social Sciences", Artificial Intelligence **267**, 1–38 (2019).

Mishra, U. A., S. Xue, Y. Chen and D. Xu, "Generative skill chaining: Long-horizon skill planning with diffusion models", in "Proc. CoRL", (2023).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning", Nature **518**, 7540, 529–533 (2015).

Molina, D., K. Kumar and S. Srivastava, "Learn and link: Learning critical regions for efficient planning", in "Proc. ICRA", (2020a).

Molina, D., K. Kumar and S. Srivastava, "Learn and link: Learning critical regions for efficient planning", (2020b).

Muise, C., S. A. McIlraith and J. Beck, “Improved non-deterministic planning by exploiting state relevance”, in “Proc. International Conference on Automated Planning and Scheduling”, (2012).

Nachum, O., S. Gu, H. Lee and S. Levine, “Near-optimal representation learning for hierarchical reinforcement learning”, in “Proc. ICLR”, (2019).

Nachum, O., S. S. Gu, H. Lee and S. Levine, “Data-efficient hierarchical reinforcement learning”, in “Proc. NeurIPS”, (2018).

Nayyar, R. K., P. Verma and S. Srivastava, “Differential assessment of black-box AI agents”, in “Proc. AAAI”, (2022).

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library”, in “Proc. NeurIPS”, (2019).

Pathak, D., C. Lu, T. Darrell, P. Isola and A. A. Efros, “Learning to control self-assembling morphologies: a study of generalization via modularity”, in “Proceedings of Advances in Neural Information Processing Systems”, (2019).

Paul, S., J. Vanbaar and A. Roy-Chowdhury, “Learning from trajectories via subgoal discovery”, in “Advances in Neural Information Processing Systems”, vol. 32 (2019).

Pineda, L., “MDP-Lib”, `https://github.com/luisenp/mdp-lib` (2014).

Pivtoraiko, M., R. A. Knepper and A. Kelly, “Differentially constrained mobile robot motion planning in state lattices”, Journal of Field Robotics **26**, 3, 308–333 (2009).

Plaku, E. and G. D. Hager, “Sampling-based motion and symbolic action planning with geometric and differential constraints”, in “In Proc. International Conference of Robotics and Automation”, (2010).

Platt Jr, R., R. Tedrake, L. Kaelbling and T. Lozano-Perez, “Belief space planning assuming maximum likelihood observations”, in “Proc. Robotics: Science and Systems”, (2010).

Rana, K., J. Haviland, S. Garg, J. Abou-Chakra, I. Reid and N. Suenderhauf, “SayPlan: Grounding large language models using 3D scene graphs for scalable robot task planning”, in “Proc. CoRL”, (2023).

Roberts, J. O., G. Mastorakis, B. Lazaruk, S. Franco, A. A. Stokes and S. Bernardini, “vPlanSim: An Open Source Graphical Interface for the Visualisation and Simulation of AI Systems”, in “Proc. ICAPS”, (2021).

Ronneberger, O., P. Fischer and T. Brox, “U-net: Convolutional networks for biomedical image segmentation”, in “Proc. MICCAI, 2015”, (2015).

Sacerdoti, E. D., “Planning in a hierarchy of abstraction spaces”, Artificial intelligence **5**, 2, 115–135 (1974).

Saribatur, Z. G., P. Schüller and T. Eiter, “Abstraction for non-ground answer set programs”, in “Proc. European Conference on Artificial Intelligence”, (2019).

Saxena, D. M., T. Kusnur and M. Likhachev, “AMRA*: Anytime multi-resolution multi-heuristic a”, in “Proc. ICRA”, (IEEE, 2022).

Seipp, J. and M. Helmert, “Counterexample-guided cartesian abstraction refinement”, in “Proc. Autonomous Agents and Multiagent Systems”, (2013).

Seipp, J. and M. Helmert, “Counterexample-guided cartesian abstraction refinement for classical planning”, Journal of Artificial Intelligence Research **62**, 535–577 (2018).

Shah, D., A. Sridhar, A. Bhorkar, N. Hirose and S. Levine, “Gnm: A general navigation model to drive any robot”, in “Proceedings of International Conference on Robotics and Automation”, (2023).

Shah, N., J. Nagpal, P. Verma and S. Srivastava, “From reals to logic and back: Inventing symbolic vocabularies, actions and models for planning from raw data”, in “arXiv preprint arXiv:2402.11871”, (2024).

Shah, N., A. Srinet and S. Srivastava, “Learning and using abstractions for robot planning”, in “ICAPS Workshop on Planning in Robotics (PlanRob)”, (2021).

Shah, N. and S. Srivastava, “An anytime hierarchical approach for stochastic task and motion planning”, in “arXiv preprint arXiv:2108.12537”, (2021).

Shah, N. and S. Srivastava, “Multi-task option learning and discovery for stochastic path planning”, arXiv preprint arXiv:2210.00068 (2022a).

Shah, N. and S. Srivastava, “Using deep learning to bootstrap abstractions for hierarchical robot planning”, in “Proc. AAMAS”, (2022b).

Shah, N. and S. Srivastava, “Hierarchical planning and learning for robots in stochastic settings using zero-shot option invention”, in “Proc. AAAI”, (2024).

Shah, N., D. K. Vasudevan, K. Kumar, P. Kamojjhala and S. Srivastava, “Anytime integrated task and motion policies for stochastic environments”, in “Proc. ICRA, 2020”, (2020).

Shah, N., P. Verma, T. Angle and S. Srivastava, “JEDAI: A system for skill-aligned explainable robot planning”, in “Proc. AAMAS”, (2022).

Shen, W., F. Trevizan and S. Thiébaux, “Learning domain-independent planning heuristics with hypergraph networks”, in “Proc. ICAPS, 2020”, (2020).

Shridhar, M., L. Manuelli and D. Fox, “Perceiver-actor: A multi-task transformer for robotic manipulation”, in “Proc. CoRL”, (2023).

Silver, D. and K. Ciosek, “Compositional planning using optimal option models”, in “Proc. ICML”, (2012).

Silver, T., A. Athalye, J. B. Tenenbaum, T. Lozano-Perez and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning", arXiv preprint arXiv:2206.10680 (2022).

Silver, T., R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. P. Kaelbling and J. Tenenbaum, "Predicate invention for bilevel planning", in "Proc. AAAI", (2023).

Silver, T., R. Chitnis, J. Tenenbaum, L. P. Kaelbling and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning", arXiv preprint arXiv:2103.00589 (2021).

Şimşek, Ö., A. P. Wolfe and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning", in "Proc. ICML", (2005).

Singh, S. P., T. Jaakkola and M. I. Jordan, "Reinforcement learning with soft state aggregation", in "Proc. Neural Information Processing Systems", (1995).

Sohn, K., H. Lee and X. Yan, "Learning structured output representation using deep conditional generative models", in "Proc. NIPS, 2015", (2015).

Speck, D., R. Mattmüller and B. Nebel, "Symbolic top-k planning", in "Proc. AAAI", (2020).

Sreedharan, S., S. Srivastava and S. Kambhampati, "Hierarchical Expertise Level Modeling for User-Specific Contrastive Explanations", in "Proc. IJCAI", (2018).

Sreedharan, S., S. Srivastava and S. Kambhampati, "Using State Abstractions to Compute Personalized Contrastive Explanations for AI Agent Behavior", Artificial Intelligence **301**, 103570 (2021).

Srivastava, B., S. Kambhampati and M. B. Do, "Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan", Artificial Intelligence **131**, 1-2, 73–134 (2001).

Srivastava, S., "Unifying Principles and Metrics for Safe and Assistive AI", in "Proc. AAAI", (2021).

Srivastava, S., E. Fang, L. Riano, R. Chitnis, S. Russell and P. Abbeel, "A modular approach to task and motion planning with an extensible planner-independent interface layer", in "Proc. International Conference of Robotics and Automation", (2014).

Srivastava, S., N. Immerman and S. Zilberstein, "Learning generalized plans using abstract counting", in "Proc. Association for the Advancement of Artificial Intelligence", (2008).

Srivastava, S., N. Immerman and S. Zilberstein, "A new representation and associated algorithms for generalized planning", Artificial Intelligence **175**, 2, 615–647 (2011).

Srivastava, S., S. Russell and A. Pinto, "Metaphysics of planning domain descriptions", in "Proc. AAAI", (2016a).

Srivastava, S., S. J. Russell and A. Pinto, "Metaphysics of planning domain descriptions.", in "Proc. Association for the Advancement of Artificial Intelligence", (2016b).

Stern, R. and B. Juba, “Efficient, safe, and probably approximately complete learning of action models”, in “Proc. IJCAI”, (2017).

Stolle, M. and D. Precup, “Learning options in reinforcement learning”, in “International Symposium on abstraction, reformulation, and approximation”, pp. 212–223 (Springer, 2002).

Sun, W., J. van den Berg and R. Alterovitz, “Stochastic extended lqr for optimization-based motion planning under uncertainty”, IEEE Transactions on Automation Science and Engineering **13**, 2, 437–447 (2016).

Sutton, R. S., D. Precup and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning”, Artificial intelligence **112**, 1-2, 181–211 (1999).

Tamar, A., G. Thomas, T. Zhang, S. Levine and P. Abbeel, “Learning from the hindsight plan—episodic MPC improvement”, in “Proc. ICRA”, (2017).

Tamar, A., Y. Wu, G. Thomas, S. Levine and P. Abbeel, “Value iteration networks”, Advances in neural information processing systems **29** (2016).

Tang, C., Q. Zhu, W. Wu, W. Huang, C. Hong and X. Niu, “Planet: Improved convolutional neural networks with image enhancement for image classification”, Mathematical Problems in Engineering **2020** (2020).

Ugur, E. and J. Piater, “Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning”, in “Proc. ICRA”, (2015).

Valmeekam, K., M. Marquez, S. Sreedharan and S. Kambhampati, “On the planning abilities of large language models–A critical investigation”, in “Proc. NeurIPS”, (2023).

Van Den Berg, J., S. Patil and R. Alterovitz, “Motion planning under uncertainty using iterative local optimization in belief space”, The International Journal of Robotics Research **31**, 11, 1263–1278 (2012).

Verma, P., S. R. Marpally and S. Srivastava, “Asking the right questions: Learning interpretable action models through query answering”, in “Proc. AAAI”, (2021).

Verma, P., S. R. Marpally and S. Srivastava, “Discovering user-interpretable capabilities of black-box planning agents”, in “Proc. KR”, (2022).

Vitus, M. P., W. Zhang and C. J. Tomlin, “A hierarchical method for stochastic motion planning in uncertain environments”, in “Proc. IROS”, (IEEE, 2012).

Vuong, Q., S. Levine, H. R. Walke, K. Pertsch, A. Singh, R. Doshi, C. Xu, J. Luo, L. Tan, D. Shah, C. Finn, M. Du, M. J. Kim, A. Khazatsky, J. H. Yang, T. Z. Zhao, K. Goldberg *et al.*, “Open X-Embodiment: Robotic learning datasets and RT-X models”, in “CoRL 2023 TGR Workshop”, (2023).

Wang, J., T. Zhang, N. Ma, Z. Li, H. Ma, F. Meng and M. Q.-H. Meng, "A survey of learning-based robot motion planning", IET Cyber-Systems and Robotics (2021).

Wang, X., "Learning planning operators by observation and practice", in "Proc. AIPS", (1994).

Wang, Z., C. R. Garrett, L. P. Kaelbling and T. Lozano-Pérez, "Active model learning and diverse action sampling for task and motion planning", in "Proc. International Conference on Intelligent Robots and Systems", (2018).

Watter, M., J. Springenberg, J. Boedecker and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images", in "Proc. NeurIPS", (2015).

Weintrop, D., A. Afzal, J. Salac, P. Francis, B. Li, D. C. Shepherd and D. Franklin, "Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices", in "Proc. CHI", (2018).

Winterer, M., C. Salomon, J. Köberle, R. Ramler and M. Schittengruber, "An Expert Review on the Applicability of Blockly for Industrial Robot Programming", in "Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)", (2020).

Wise, M., M. Ferguson, D. King, E. Diehr and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications", in "Workshop on Autonomous Mobile Service Robots", (2016).

Yang, F., D. Lyu, B. Liu and S. Gustafson, "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making", in "Proc. IJCAI", (2018).

Yang, Q., K. Wu and Y. Jiang, "Learning action models from plan examples using weighted MAX-SAT", AIJ **171**, 2-3, 107–143 (2007).

Yu, W., N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa and F. Xia, "Language to rewards for robotic skill synthesis", in "Proc. CoRL", (2023).

Zhang, C., J. Huh and D. D. Lee, "Learning implicit sampling distributions for motion planning", in "Proc. IROS, 2018", (2018).

Zhuo, H. H. and S. Kambhampati, "Action-model acquisition from noisy plan traces", in "Proc. IJCAI", (2013).