# Learned Functions for Perceptually Informed Robot Navigation

by

Martina Katherine Stadler

B.S., Massachusetts Institute of Technology (2018)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020



Author .....
Department of Aeronautics and Astronautics
August 18, 2020

Certified by .....
Nicholas Roy
Professor, Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Zoltan Spakovszky
Professor, Aeronautics and Astronautics
Chair, Graduate Program Committee

# Learned Functions for Perceptually Informed Robot Navigation

by

Martina Katherine Stadler

Submitted to the Department of Aeronautics and Astronautics
on August 18, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

## Abstract

While existing robotic systems predominantly rely on geometric information to inform robot navigation, non-geometric information, such as object-level maps and overhead imagery, provide rich navigation cues that can be used to inform intelligent navigation behaviors. However, it is not obvious how non-geometric navigation cues should be incorporated into existing robot motion planning pipelines.

This thesis presents two novel methods that use learning to incorporate non-geometric information into classical planning techniques for robot navigation. First, we present Learned Sampling Distributions, a novel method for learning a sampling distribution based on local hybrid geometric and object-level maps to inform a sampling-based motion planner for navigation in unknown environments. Our approach uses expert demonstrations to learn a probability distribution that places high probability in regions of the environment that are likely to be on optimal paths to the goal, like hallways and doorways in an office environment, and results in up to a 2.7x increase in the probability of finding a plan for a resource-constrained agent when compared to a baseline planner. Second, we present Perceptually Informed Abstractions, a novel method for hierarchical planning at long length scales that learns properties of abstract actions for use in a risk-aware hierarchical discrete planner, conditioned on low-resolution overhead images. We also present a preliminary analysis of the approach in a simulated toy environment.

Thesis Supervisor: Nicholas Roy
Title: Professor, Aeronautics and Astronautics

# Acknowledgments

First, I would like to thank my parents, Patti and Scott, for teaching me the value of an education and for supporting me relentlessly throughout my studies. Dad, thank you for always showing me your pet projects, and for encouraging me to set my sights on MIT. Mom, thank you for not only teaching me, but showing me the value of education by going back to school (and getting a PhD!), all while raising your kids. Thank you Berna and JJ, for contributing to a home where work and play were equally important. Thank you Andrew, for your endless supply of support, encouragement, and patience, and for always knowing how to make me laugh. While I've certainly appreciated your many contributions to this thesis since becoming your home office mate, I do hope that none of them have quite made it into this document.

I would also like to thank the people who have shaped my academic career. Thank you Nick Roy, for introducing me to the world of robotics research as an undergraduate student, and for continuing to support me throughout my graduate career. Thank you to all the members of the Robust Robotics Group for your insights. But especially, thank you Kyel Ok, for originally encouraging me to pursue research and graduate studies in the RRG. And finally, thank you Katherine Liu, for taking me on both as a UROP and a coauthor. You've taught me most of the things I know about how to be a researcher, and for that I will forever be indebted to you.

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-17-2-0181. Their support is gratefully acknowledged.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

In recent years, there has been significant interest in transitioning robots from controlled laboratory and factory environments to real-world, uncontrolled environments. Robust, efficient robot navigation is one key capability for deploying robots in the real world, as basic mobility is a necessary component of many robotic systems, from search-and-rescue vehicles to assistive care robots. Unfortunately, many real-world robots are confined to simple navigation paradigms that rely on dense geometric mapping alone to avoid collisions with the environment and to inform intelligent navigation behaviors. In practice, these pipelines often make assumptions that lead to myopic real-world planning, like assuming that unknown space is free, and that Euclidean distance heuristics are good approximations of plan costs in complex environments.

While these assumptions are sufficient for planning in small, known environments, they lead to poor navigation results in structured, unknown environments. For example, consider the two scenarios in Figure 1-1. We assume that a ground vehicle equipped with a depth camera begins a navigation task at the green circle. The robot has no map of the environment a priori, but has a simultaneous localization and mapping (SLAM) system that localizes the robot and builds a map of the world online (e.g., Mur-Artal and Tardós [60], Rosinol et al. [74]). In (a), the robot is in a

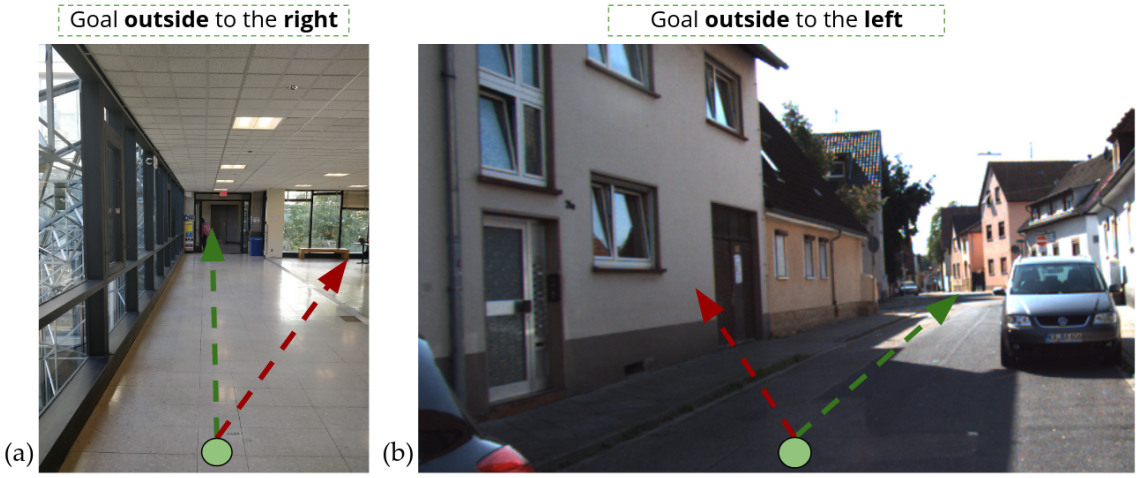


Figure 1-1: **Two examples of non-Euclidean navigation behaviors in structured, unknown environments.** Consider a ground vehicle equipped with a depth camera that begins a navigation task at the green circle in an unknown environment. The robot uses a SLAM system to localize and build a partial map of the environment online. (a) The robot is in a hallway and is tasked with navigating to a goal that is outside and to the right. Instead of greedily moving towards the windows to minimize the Euclidean distance to the goal (red dashed arrow), the robot recognizes an exit sign and follows the hallway to exit the building (green dashed arrow). (b) The robot is on a road and is tasked with navigating to a goal that is outside and to the left. Instead of greedily attempting to reach the goal by navigating through the building (red dashed arrow), the robot follows the road until it reaches the nearest intersection (green dashed arrow). The image in (b) is from the KITTI dataset [25].

hallway and is tasked with reaching a goal that is outside and to the right. In this example, the optimal navigation behavior is to navigate down the hallway towards the door and exit sign, which indicates a method of egress, to exit the building and then turn right to navigate towards the goal. However, due to limitations in depth sensing (e.g., range and field of view), the robot has not geometrically mapped the far wall of the building at the start of the task, and greedily navigates to the wall in an attempt to minimize the Euclidean distance to the goal. The robot must navigate to and map the entire wall before realizing that it is in a bug trap, and only then exits through the door at the end of the hallway. In (b), the robot is tasked with navigating to a goal that is outside and to the left. The optimal navigation behavior is to navigate down the road and turn left at a nearby intersection. However, because the robot assumes that unknown space is free, it greedily moves towards the building

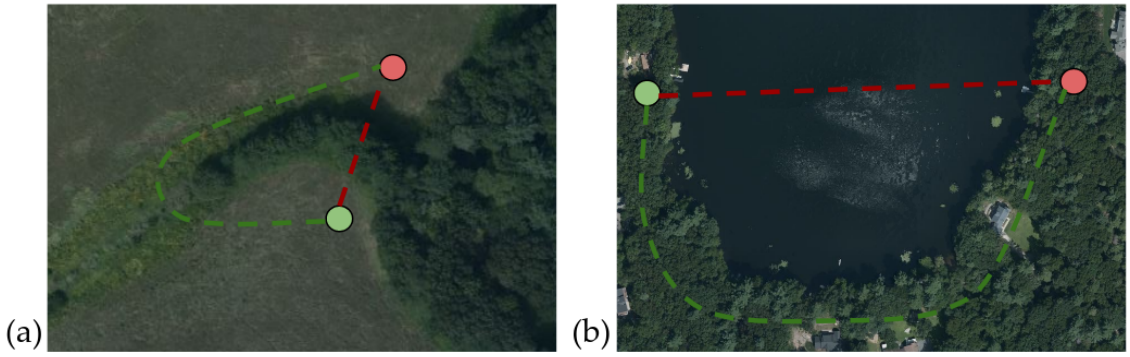


Figure 1-2: **Two examples of non-Euclidean navigation behaviors in large, known environments.** Consider a ground vehicle that begins a navigation task at the green circle in a large, known environment. (a) The Euclidean distance heuristic encourages the robot to explore a forest region, which is expensive to plan and navigate in (red dashed line), despite the existence of a lower cost, simpler path that navigates around the forest (green dashed line). (b) The Euclidean distance heuristic encourages the robot to consider plans through an untraversable lake (red dashed line), instead of focusing computation on generating a path through the complex forest environment (green dashed line). Images collected from Bing Maps [58].

in an attempt to minimize the Euclidean distance to the goal, and only continues down the road once the building has been fully mapped.

The failure modes of assuming a Euclidean distance heuristic are not unique to navigation in unknown environments. They also occur in known environments over long length scales, where a poor choice of heuristic can cause planners to waste computation considering high cost plans. For example, consider the two scenarios in Figure 1-2, in which a ground vehicle equipped with a known map of the environment attempts to navigate from the green circle to the red circle in a large, outdoor environment. In (a), the Euclidean distance heuristic encourages the robot to navigate through a forest, although the complex geometry of the forest (i.e., trees) makes it unlikely to be part of a low-cost trajectory. In (b), the Euclidean distance heuristic encourages the ground vehicle to attempt to navigate through a lake, and the robot wastes computation attempting to find a valid trajectory that goes through the water, instead of focusing computation on solving the challenging navigation task through the forest.

## 1.2 Non-Geometric Navigation Behaviors

Intuitively, humans do not struggle with navigation tasks in the ways that robots do. It is unlikely that a human would attempt to navigate through a wall in a building, or navigate into a building when trying to navigate between outdoor locations. Similarly, it is unlikely that a human would attempt to navigate through a forest or a lake, if given another option. However, these intuitive strategies – avoiding walls and buildings, using doors and exit signs to inform egress, and quickly identifying high cost or untraversable areas for navigation – rely on a non-geometric understanding of the world. In this thesis, we argue that reasoning about navigation behaviors using non-geometric information is necessary to enable intelligent navigation in structured environments.

This claim is partially supported by real-world intelligent navigators. It has been shown that humans and other mammals have specific hardware in the brain that encodes non-geometric cues for navigation. Rats have specialized cells that identify absolute orientations, specific locations, relative translations, and borders of closed environments [18]. Humans encode navigable regions in the occipital place area, a specialized region of the visual cortex, even while completing tasks unrelated to navigation [66]. Behaviorally, expert human taxi drivers showed improved performance on place recognition tasks from images as compared to novice taxi drivers, but did not show improved performance on geometry-based tasks, such as map drawing and place location on maps [16]. While mammal navigation behaviors are not fully understood, these examples indicate that non-geometric information may be important to enabling intelligent navigation behaviors.

However, many advancements in robot motion planning have largely been confined to improving navigation results in fully-known, noiseless geometric maps in simulation and in carefully controlled laboratory environments (e.g., [35, 36, 59]). For example, Ichter et al. [35] learned a sampling distribution for sampling-based motion planning which placed samples in regions of the environment that were likely to be on optimal paths to the goal, like narrow passageways in a small maze, but only considered

planning in fully known maps. While this and other techniques effectively exploit fully observed geometric features of the environment to improve motion planning outcomes, they do not incorporate non-geometric information or environmental uncertainty into intelligent navigation behaviors.

There are a number of challenges to incorporating non-geometric information into robot motion planning paradigms. First, it is not clear what kind of non-geometric information should be used to inform intelligent planning. In the toy example in Figure 1-1 alone, we identified objects (i.e., exit signs and doors), non-object semantic classes (i.e., hallways, buildings, and roads), and semantic context (i.e, the fact that the goal was outdoors) as important signals when defining navigation behaviors. Second, while we know that intelligent agents store many kinds of non-geometric information, it is not clear when and how these agents use different sources of information to make planning decisions.

In this thesis, we explore various representations of geometric and non-geometric information for use in intelligent navigation pipelines. In some cases, we represent non-geometric information explicitly. We use object-level maps, generated from a metrically accurate object-level SLAM system based on 2D object detections [65], to explicitly represent windows, doors, and exit signs in 2D metric space. Additionally, we use *semantic context* in the form of a boolean indicator to explicitly indicate whether or not the goal of the robot is indoors or outdoors. In this thesis, we will refer to explicit non-geometric information as *semantic information*. In other cases, we represent non-geometric information implicitly. We use overhead images to implicitly represent non-object semantic classes, such as buildings and roads, and we use partial occupancy maps, built online, to implicitly identify hallways based on partial geometric structure. In this thesis, we will refer to implicit geometric and non-geometric information as *perceptual abstractions*. By using various representations of non-geometric information, we are able to encode the signals necessary to inform intelligent robot navigation.

Along with encoding semantic information and perceptual abstractions, we need to enable robots to *use* non-geometric information when making planning decisions.

While some approaches have attempted to explicitly write down desired navigation behaviors relative to semantic and perceptual cues, hand-crafted behaviors are unlikely to scale to different robots or complex environments [72, 18]. For example, reconsider the hallway in Figure 1-1a. In the original ground vehicle formulation, the optimal navigation strategy exited the building through the door at the end of the hallway. However, if the door at the end of the hallway no longer had an exit sign above it, navigating to the door may not be part of an optimal planning strategy. Similarly, if we now assume that the robot is a quadrotor, and that the windows on the far wall are open, the red path is likely to be on the optimal path to the goal. Because accurately interpreting semantic information and perceptual abstractions to inform planning requires complex reasoning about the state of the environment and the state of the agent, hand-coding behaviors for navigation in complex environments will be tedious, if not intractable.

To avoid explicitly defining semantically- and perceptually-informed navigation behaviors, we propose learning functions that encode these behaviors based on example optimal trajectories. We use off-the-shelf optimal planners (e.g., [84]) in simulated fully known environments that model geometric and non-geometric information to generate datasets of correlated optimal trajectories, geometric information, and non-geometric information. Then, we combine these datasets with neural networks to learn robot navigation behaviors from a subset of the available geometric and non-geometric information. Specifically, we encode the learned navigation behaviors in functions that can be used in classical planning pipelines.

### 1.2.1 Learned Sampling Distributions

The first contribution of this thesis is Learned Sampling Distributions for Efficient Planning in Hybrid Geometric and Object-Level Representations, a novel method that uses partially observed geometric maps, object-level maps, and semantic context to learn a sampling distribution and cost function for use in a sampling-based motion planner in unknown environments. By biasing sampling to areas of the environment that are likely to be on optimal paths to the goal, we demonstrate that the learned

planner is more likely to exhibit semantically-informed behaviors, such as staying in hallways, using doors to enter rooms, and using doors with exit signs to exit buildings. We demonstrate that our approach is up to 2.7 times more likely to find a valid plan than a sampling-based motion planner using a uniform sampling distribution and a Euclidean distance cost function for a resource-constrained agent in a simulated university floorplan environment. We also demonstrate that our approach can lead to less costly plans than the baseline approach in some environments. Finally, we demonstrate promising qualitative results on a 1/10th scale RC car navigating online in a building at MIT.

### 1.2.2 Perceptually Informed Abstractions

The second contribution of this thesis is Perceptually Informed Abstractions for Efficient Robot Navigation, a preliminary method that uses coarse overhead imagery to learn cost and traversability functions to inform hierarchical navigation in long length scale environments. By using overhead imagery to approximate the cost and feasibility of low-resolution abstract actions, we enable an agent to quickly identify subsets of high-resolution plans that are likely to be low cost, while avoiding high-resolution planning in areas of the environment that are likely to contain high cost solutions. We demonstrate preliminary results of our approach in a toy outdoor environment.

## 1.3 Thesis Overview

In this thesis, we present novel methods for using learning to incorporate non-geometric information into classical planning techniques. In Chapter 2, we introduce the planning problem and discuss existing work in the field of informed robot motion planning. In Chapter 3, we introduce Learned Sampling Distributions, an approach that uses hybrid geometric and object-level maps to learn a sampling distribution for sampling-based motion planning in partially observed environments. In Chapter 4, we present the results of using the Learned Sampling Distributions method in a simulated university floorplan environment, and demonstrate promising results on a 1/10th scale

RC car navigating in a building at MIT. In Chapter 5, we introduce Perceptually Informed Abstractions, an approach that uses overhead images to learn a cost function and a traversability function for abstract actions in a hierarchical planner. We also present a preliminary analysis of the method in a simulated toy outdoor environment. Finally, we conclude and discuss potential avenues for future work in Chapter 6.

# Chapter 2

# Related Work

In this chapter, we formalize the continuous and discrete optimal motion planning problems. We review historical and state-of-the-art motion planning algorithms, and motivate the use of non-geometric information to inform planning decisions for robot navigation.

## 2.1 The Continuous Optimal Planning Problem

Consider a robot with state $x$ defined in some continuous configuration space $\mathcal{X} \in \mathbb{R}^d$, where $d$ is the dimensionality of the state. The goal of the motion planning problem is to find a valid (i.e., non-colliding and dynamically feasible) plan that takes the robot from an initial state $x_s$ to a goal state $x_g$, where $x_s, x_g \in \mathcal{X}$. We assume that each state can be classified as either valid or invalid, and that there exists a known function that determines the validity of each state, $f : \mathcal{X} \to \{0, 1\}$. We denote a plan $\rho$ as a continuous mapping through $\mathcal{X}$, i.e., $\rho : [0, 1] \to \mathbb{R}^d$, and assume that an additive scalar cost function, $c(\rho) \in \mathbb{R}$, is given. Formally, we define the optimal planning problem as the minimization of $c(\rho)$ subject to constraints,

$$\begin{aligned} \rho^* &= \arg\min_{\rho} \ c(\rho) \\ \text{s.t.} \quad & I_\rho(\rho^*) = 1 \\ & \rho^*(0) = x_s, \rho^*(1) = x_g, \end{aligned} \tag{2.1}$$

where $I_\rho(\rho)$ is an indicator function that evaluates the validity of $\rho$,

$$I_\rho(\rho) = \begin{cases} 1 & \text{if } f(\rho(t)) \ \ \forall t \in [0,1] \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

In general, this optimization can be high-dimensional and non-convex. While the complexity class of the optimization is dependent on the problem input (e.g., the representation of the robot and the representation of the obstacles in the environment), for some general input classes, such as the general motion planning problem (i.e., the piano mover's problem), the optimization has been proven to be PSPACE-complete [73, 14]. While there exist continuous planners that solve Equation 2.1 exactly, these planners rely on clever decompositions of the environment that require obstacles to be defined in restrictive ways (e.g., analytically defined polygons) [51]. While it may be possible to generate analytic representations of an environment offline or in simulation, this representation does not support real-world robot navigation in complex environments using noisy sensors. Instead, we turn to methods that approximate solutions to Equation 2.1 to solve real-world navigation problems.

In practice, there are two main methods for approximating the optimization in Equation 2.1: *local* methods (e.g., trajectory optimization [43, 101]), which use gradient-based techniques to find locally optimal navigation behaviors, and *global* methods (e.g., graph-based methods, sampling-based methods), which use discrete approximations of the configuration space to enable efficient search for approximate global solutions to the optimization. In this thesis, we focus on global methods due to our interest in generating plans in structured and long length scale environments, where local methods are likely to suffer from local minima.

## 2.2 The Discrete Optimal Planning Problem

To reduce some of the intractability of the optimization in Equation 2.1, we can approximate continuous plans as sequences of discrete *primitive* actions $a \in \mathcal{A}$ that can be executed by the robot in the environment. Formally, we define a sequence of primitive actions as a primitive plan, $p = a_0 \circ a_1 \circ ... \circ a_{n-1}$, where $\circ$ is the action concatenation operator, $n$ is the number of actions in the plan, and $p \in \mathcal{P}$, where $\mathcal{P}$ is the space of all possible primitive plans. Additionally, we assume a known partition function that maps every $\mathcal{A}$ to its validity, $f : \mathcal{A} \to \{0, 1\}$, and a known scalar cost function, $c(p) \in \mathbb{R}$. Formally, we define the discrete optimal planning problem,

$$\begin{aligned} & p^* = \arg\min_{p} \ c(p) \\ \text{s.t.} \quad & I_p(p^*) = 1 \\ & p^*(0) = x_s, p^*(1) = x_g, \end{aligned} \tag{2.3}$$

where $I_p(p)$ is a function that evaluates the validity of a primitive plan,

$$I_p(p) = \begin{cases} 1 & \text{if } f(a_t) \ \forall t \in \{0, 1, ..., n-1\} \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

While committing to a finite action set $\mathcal{A}$ guarantees that, for some problems, the approximation will not be able to represent optimal and in some cases valid solutions to Equation 2.1, in the limit of an infinitely expressive $\mathcal{A}$, $\rho^*$ can be well-approximated by $p^*$.

In practice, there are two key challenges involved in implementing discrete optimal planning for real-world robot navigation. First, an approximation scheme that converts a continuous state space into a discrete model of actions $a$ that is amenable to planning must be defined; we call this process *model construction*. Second, an algorithm that uses the discrete model to search for solutions $p^*$ to Equation 2.3 must be defined; we call this process *search*. We discuss model construction and search in

detail in Sections 2.3 and 2.4.

## 2.3 Model Construction

The goal of model construction is to generate an approximate model of the continuous planning problem that is both *representative* of the original planning problem (i.e., that maintains elements of the original planning problem that inform good navigation behaviors), and that is *tractable* to build and search in. Throughout this section, we analyze the representativeness of a model using two metrics, model *completeness*, or whether a valid plan that satisfies the constraints in Equation 2.3 exists in the model, and model *optimality*, or whether a valid plan that minimizes the objective function in Equation 2.3 exists in the model. We also use two metrics to quantify the tractability of different models, namely by describing the size of the plan space, $\mathcal{P}$, induced by the model. First, we consider the model *branching factor*, or the number of actions that can be taken by the agent from a given state in the model. Second, we consider the model *depth*, or the number of actions in a solution path, given the model. In general, the complexity of the search process goes as $\mathcal{O}(\text{branching factor}^{\text{depth}})$. Unfortunately, this results in a tension between representative and tractable models. To enable tractable planning, we would like to develop search algorithms that have a small depth and branching factor; however, these models require generating a compact, and often less expressive model of the environment, leading to decreased model representativeness. In this section, we consider a number of approaches that cleverly trade off between model representativeness and tractability.

### 2.3.1 Configuration Space Discretization

The first model we consider to optimize Equation 2.3 is configuration space discretization. The key insight of state discretization is that continuous states that are close to each other in configuration space are likely to have similar properties, and can therefore be considered jointly when searching for plans. Formally, a state discretization is generated by decomposing a continuous configuration space into a finite set of

discrete states, $x_d \in \mathcal{X}_d$. The most common methods of state discretization assume a constant-resolution discretization, as in standard occupancy maps [75, 57], or a hierarchy of constant-resolution discretizations, as in octree-based methods [32]. Then, the set of discrete actions, $a_d \in \mathcal{A}_d$, is recovered by making assumptions about valid transitions between configurations. For holonomic agents, or agents using a low-level controller that approximates holonomic behaviour, it is commonly assumed that actions cause robots to transition between directly adjacent states, or neighbors, in the discretized configuration space. For agents with more complex dynamics, methods such as state latices [70] can be used to identify an alternative set of actions that obey dynamic feasibility constraints.

To analyze the model, we first consider its representativeness. Models based on discretized configuration spaces are only capable of representing plans at the resolution of the state discretization. For problems where the discretization resolution is well fit to the configuration space, discretized models can efficiently represent low cost solutions to challenging planning problems. However, poorly discretized models can lead to computational inefficiency, or to models that are unable to represent optimal or even feasible plans. With that being said, discretized state models represent all possible plans that can be represented at a given resolution, and are therefore called *resolution-complete*. Similarly, because discretized models represent all plans at a given resolution, they necessarily represent the lowest cost plan at a given resolution, and are therefore called *resolution-optimal*.

Next, we consider the model's tractability. The branching factor of the model is defined by the number of states adjacent to every state in the discretized configuration space, and is most commonly a small constant. However, the model depth, or the number of actions in a plan, is linearly dependent on the resolution of the discretization (i.e., a plan at half the resolution could be represented in half the number of actions). However, because of the exponential relationship between plan depth and the complexity of the search process, linearly higher resolution models induce exponentially more expensive search problems. In practice, discretized state models can be intractable to build and search in for planning problems in structured and

long length scale environments.

### 2.3.2 Graphs for Robot Motion Planning

To avoid the computational intractability of discretized state methods, graph-based models for motion planning were introduced. The key insight of graph-based models is that the length of each action in a plan should be based on the complexity of the local configuration space. In areas of the configuration space with significant structure, the robot should consider short actions that can capture the features of the configuration space, but in regions of the environment with little structure, longer actions are sufficient to represent possible traversals in the environment.

Formally, we define a planning graph $G$ as a collection of vertices $V$ and edges $E$, $G = \{V, E\}$, where $V = \{v_0, v_1, ..., v_n\}$, and $E = \{e_0, e_1, ..., e_m\}$. For the planning problem, each vertex $v$ represents a state $x$ in the configuration space, and each edge $e$ represents a valid action that takes the robot from some state $x_i$ to a different state $x_j$. We assume known validity functions over vertices and edges, $f_v : V \rightarrow \{0, 1\}$ and $f_e : E \rightarrow \{0, 1\}$, and we assume that all vertices and edges in $G$ are valid (i.e., $f_v(v) = 1 \; \forall v \in V$ and $f_e(e) = 1 \; \forall e \in E$). Then, we can define a valid trajectory $\mathcal{T}$ as a directed, acyclic subgraph of $G$. With this model, we can rewrite the optimization in Equation 2.3 as a search over subgraphs of $G$:

$$\begin{aligned} & \mathcal{T}^* = \arg\min_{\mathcal{T}} \boldsymbol{C}(\mathcal{T}) \\ \text{s.t.} \quad & \mathcal{T}^* = \{V^*, E^*\}, b(e_0) = x_s, t(e_m) = x_g, \\ & v \neq v' \quad \forall v, v' \in V^* \\ & t(e_i) = b(e_{i+1}) \; \forall i \in \{1, 2, ..., m-1\} \\ & V^* \in V, E^* \in E, \end{aligned} \tag{2.5}$$

where $\boldsymbol{C}(\mathcal{T}) \in \mathbb{R}$ is a cost function defined over trajectories, $t(e)$ indicates the terminal node of edge $e$, and $b(e)$ indicates the start node of edge $e$.

To analyze graph-based methods, we first consider their representativeness. Early

approaches to planning graph construction focused on computing minimal graphs of the environment based on the geometric properties of valid and invalid robot configurations by using tools from computational geometry, such as visibility graphs [53] and Voronoi diagrams [86], to determine the connectivity of the configuration space. By directly reasoning about the geometry of the environment, these approaches were guaranteed to represent all minimum-cost paths in an environment, under some assumptions (e.g., the robot was often assumed to be holonomic). Therefore, these planning graphs were guaranteed to represent all possible optimal traversals in the environment, and were both complete and optimal.

Next, we consider the tractability of graph-based methods. The branching factor of graph-based models was dependent on the exact graph construction technique, but was normally a small value correlated with the number of obstacles in the local region of the environment. However, graph-based methods were able to significantly reduce the depth of planning solutions using edges. Specifically, long paths through open areas of the environment were compressed into single actions (i.e., edges), allowing the number of actions in a path to be correlated with the complexity of the environment, instead of the absolute length of the path.

Unfortunately, computing geometric properties of the configuration space can be expensive, especially in structured and long length scale environments. Instead, we turn to sampling-based graphs, which combine sampling and constant-time collision checking to efficiently generate planning graphs in complex environments.

### 2.3.3 Sampling-Based Graphs

To avoid the inefficiencies of direct graph construction based on configuration space geometry, sampling-based graph construction techniques were introduced [42]. The key insight in sampling-based models is that determining whether or not a given configuration or path is in collision can be more efficient than explicitly representing and reasoning about obstacles in a configuration space.

Planning graphs for sampling-based motion planning are generated by sampling and deterministically connecting states in the configuration space. Specifically, states

$x$ are randomly sampled from the continuous configuration space and considered for representation as candidate graph vertices $v$ according to a user-specified sampling distribution $\Phi(x;\Gamma)$, where $\Gamma$ are optional additional inputs to the distribution; one common choice of sampling distribution is a uniform sampling distribution [42, 50, 47]. Then, valid candidate vertices are considered for addition to the graph using a deterministic strategy. While methods for candidate vertex addition vary (in graph-based algorithms, candidate vertices are added directly to the graph; in tree-based algorithms, the tree is expanded in the direction of the candidate vertex) [51], each results in vertices distributed according to $\Phi(x;\Gamma)$ in non-colliding regions of the configuration space [42, 50].

Once vertices have been added to the planning graph, edges are deterministically added to the graph using an edge addition strategy; common strategies include connecting each vertex to its $k$-nearest neighbors in search space, connecting each vertex to all neighbors within a certain radius, or connecting each vertex to its $k$ lowest-cost neighbors [41], provided that the resulting edge represents a valid traversal in the environment. A traversal is considered valid if it does not violate any planning constraints, such as occupancy constraints and kinematic constraints. Two special cases of the $k$ neighbors strategy are the nearest neighbor tree-based strategy, where each vertex is connected to its single nearest neighbor [50], and the lowest-cost neighbor tree-based strategy, where each vertex is connected to its single lowest-cost neighbor [41]. The process of constructing and searching a sampled graph is referred to as Sampling-Based Motion Planning (SBMP). While a number of SBMP algorithms have been proposed, most are based on two prominent algorithms: Probabilistic Roadmaps (PRMs) [42] and Rapidly-Exploring Random Trees (RRTs) [50], and their optimal counterparts, PRM*s and RRT*s [41]. We refer interested readers to the respective papers for additional information.

To analyze sampling-based models, we first consider their representativeness. Because the models do not guarantee that any given vertex will be added to the graph in finite time, sampling-based models do not admit exact completeness or optimality guarantees. However, by using the theory of random geometric graphs, they do ad-

mit asymptotic guarantees, that is, guarantees of plan existence and quality as the number of samples in the graph tends to infinity. Specifically, when the sampling distribution $\Phi(x;\Gamma)$ has a non-zero probability of sampling each state $x$ in the configuration space (i.e., $\Phi(x;\Gamma) > 0 \; \forall x \in \mathcal{X}$), sampling-based models are probabilistically complete [42, 50], that is, in the limit of sampling an infinite number of candidate vertices, a sampling-based model will represent a valid plan, if one exists. Additionally, some sampling-based motion planners with intelligent edge addition strategies are asymptotically optimal [41], that is, in the limit of sampling an infinite number of candidate vertices, the sampling-based model will capture the optimal solution to the planning problem, if a solution exists.

Second, we consider the model's tractability. Because sampling-based methods build graphs, they generate similarly-sized plan spaces to traditional graph-based models. However, the sampling-based graph construction process is much less expensive than other graph construction methods. While other graph-based methods require careful computations of properties of the configuration space to generate representative graphs, sampling-based graphs rely on only randomness and inexpensive collision checking to capture the connectivity of the configuration space. This process is much less expensive than decomposing the configuration space based on geometry.

## 2.4 Search

In the following section, we discuss search methods, which use the discrete models discussed in Section 2.3 to generate solutions to the optimization in Equation 2.3. While a number of search algorithms exist, we choose to highlight forward search algorithms due to their generality and ubiquity in the planning literature.

### 2.4.1 Forward Search

The goal of the forward search algorithm is to find a valid plan from a start state to a goal state in a discrete model of the environment. Planning begins by adding the start state to a priority queue and instantiating a graph of candidate solution

paths. Then, at every iteration, a state is removed from the queue and added to the solution graph. If the state is the goal state, planning terminates. Otherwise, the state is expanded. In the expansion process, the search algorithm applies the set of valid actions at the current state to generate new states, which are added to the priority queue, if they are not already on the queue or in the solution graph. This process repeats until the goal state is found, or until no states are left on the queue, indicating that no valid plan exists.

One key to efficient search is the efficient prioritization of state expansions. For example, imagine a robot navigating between adjacent offices in an office building. The robot only needs to consider discrete states and actions between the two offices, and does not need to consider states and actions in hallways on the opposing side of the building, or on other floors of the building. In fact, considering these states and actions would likely be prohibitively expensive, given the exponential relationship between plan length and the complexity of the planning problem.

While the efficient prioritization of state expansion is important to ensuring computational tractability for forward search, it is not obvious how discrete states should be prioritized when building planning graphs. The study of prioritization schemes is centered around heuristics, which approximate the usefulness of candidate vertices for the progression of the search process as a real-valued number, or heuristic $h \in \mathbb{R}$. While there is a rich literature of heuristics for planning [51, 69, 68], the most common heuristics score vertices based on cost-to-come, the cost of traversing from the start vertex to the candidate vertex [17, 71], cost-to-go, the approximated cost of traversing from the candidate vertex to the goal vertex, or a weighted combination of cost-to-come and cost-to-go [29]. A selection of task-specific navigation heuristics are considered in Section 2.6.2.

While a variety of heuristics have been successfully deployed for robot motion planning, one particular class of heuristics, admissible consistent heuristics, have been particularly influential in the planning literature. Admissible heuristics do not overestimate the costs of plans, and consistent heuristics do not disproportionately overestimate or underestimate the cost of any one plan relative to other plans con-

sidered by the search algorithm. When an admissible, consistent heuristic is used to prioritize state expansion for forward search, the forward search algorithm, which we call A*, is guaranteed to find the lowest-cost solution in the discrete model, provided that a solution exists [29].

## 2.5 The Hierarchical Planning Problem

In Section 2.3, we discussed a number of discrete models for solving the optimization in Equation 2.3. However, for some high-dimensional and long-length scale environments, even compact discrete models of an environment are insufficient to guarantee planning tractability. Hierarchical planning algorithms increase the tractability of motion planning by representing the action space at various resolutions. Specifically, hierarchical models represent the space of possible plans using *abstract* actions $\mathbf{a}$, or actions based on simplified models of the planning problem that do not necessarily capture all planning constraints. Search proceeds using abstract actions to generate abstract plans $\mathbf{p}$ which approximate solutions to the original planning problem, where $\mathbf{p} = \mathbf{a}_0 \circ \mathbf{a}_1 \circ ... \circ \mathbf{a}_{n-1}$[1]. Formally, we can write the abstract planning problem as a minimization subject to constraints:

$$\begin{aligned} & \mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_{t=0}^{|\mathbf{p}|-1} \mathbf{c}(\mathbf{a}_t) \\ \text{s.t.} \quad & I_{\mathbf{p}}(\mathbf{p}) = 1 \\ & \mathbf{p}^*(0) = x_s, \mathbf{p}^*(1) = x_g, \end{aligned} \tag{2.6}$$

where $\mathbf{c}(\mathbf{a})$ and $\mathbf{f}(\mathbf{a})$ are abstract cost and validity functions defined over the abstract action space, and $I_{\mathbf{p}}(\mathbf{p})$ is a function that evaluates the validity of an abstract plan,

$$I_{\mathbf{p}}(\mathbf{p}) = \begin{cases} 1 & \text{if } \mathbf{f}(\mathbf{a}_t)\ \forall t \in \{0, 1, ..., |\mathbf{p}| - 1\} \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

[1]Throughout this work, we use italic type to denote primitive variables and functions, and bold type to denote abstract variables and functions.

In a slight abuse of notation, we overload the abstract cost function to simplify the notation for the cost of an abstract plan, $\mathbf{c}(\mathbf{p}) = \sum_{t=0}^{|\mathbf{p}|-1} \mathbf{c}(\mathbf{a}_t)$. Once low cost abstract plans are found, hierarchical planners use properties of $\mathbf{p}^*$ to guide the search for a primitive plan $p$ in the original configuration space, $\mathcal{X}$ (e.g., by solving Equation 2.3, but over a reduced configuration space that only includes states consistent with $\mathbf{p}$). We call a primitive plan guided by an abstract plan a primitive *refinement*.

While hierarchical planning algorithms vary significantly by application and domain, they can often be characterized by the type of decomposition used to generate abstract states and actions. In the following sections, we briefly review two types of decompositions for abstract states and actions: fact-based decompositions and metric-based decompositions.

### 2.5.1 Fact-Based Hierarchical Planning

In fact-based planners, abstract states are defined as sets of primitive states in which particular facts about the world are true. For example, consider a robot cleaning two plates in a sink. The state of the world could be represented based on which items in the sink are clean and dirty. Then, abstract actions could be defined by preconditions – facts about the world that must hold true for the abstract action to be executed – and postconditions – facts about the world that must hold true after the abstract action is executed. For example, one abstract action could be to clean a plate; its preconditions could be that the robot is at the sink, and the plate in question is dirty, and its postconditions could be that the robot is at the sink, and the plate in question is clean. Importantly, the abstract action does not capture the complete state of each item in the sink – for example, one of the plates may be dirty, small, and in the left corner of the sink, while the other plate may be dirty, large, and in the right corner of the sink – and it does not capture the primitive actions necessary to complete the abstract action (e.g., pick up the plate, put soap on the sponge, clean the plate with the sponge). However, when cleverly designed, the abstract representation can be sufficient to generate an abstract plan (first clean plate 1, then clean plate 2) that can be refined into a primitive plan for cleaning the items in the sink. In general, this

technique is more efficient than directly attempting to search for a primitive plan in the full configuration space.

One of the earliest implementations of a fact-based hierarchical planner was the STanford Research Institute Problem Solver (STRIPS) planner, which was used to generate plans for the Shakey robot [21]. The STRIPS planner defined abstract states of the world as collections of facts about the environment and the state of the robot. Abstract states were modified using abstract actions, also called operators, defined using preconditions and postconditions on the facts of the world. To plan using STRIPS, a sequence of actions was applied to the initial state of the world; planning terminated once a goal state was reached. Notably, STRIPS did not define primitive plans associated with each abstract action; instead, it was assumed that if an abstract action's preconditions and postconditions were satisfied in an abstract plan, a valid high-resolution plan refinement of the abstract action could be generated using a high-resolution planner. This hierarchical planning approach allowed the robot to generate abstract plans in abstract space, then use a motion planner to execute individual abstract actions in the abstract plan by solving a series of smaller optimizations over a reduced plan space. Fact-based planners have been extended to a variety of domains, including efficient suboptimal task and motion planning by assuming the reversibility of abstract actions with uncertain refinements [40], and task and motion planning in unknown environments while explicitly reasoning about uncertainty [46].

### 2.5.2 Metric-Based Hierarchical Planning

While fact-based hierarchical planners have greatly increased the computational tractability of complex planning problems, they can be overly complicated for the navigation problem, where a compact representation of important information for planning is already known: maps, which store metrically-correlated facts about the environment (e.g., occupancy maps). A number of approaches have been proposed to extend metric maps to the hierarchical planning setting. One common approach to metric hierarchical planning defines abstract states as grid cells in a low-resolution representation of

the environment and abstract actions as traversals through abstract grid cells. Early approaches used hand-defined metrics to approximate the costs of abstract actions; for example, Thorpe and Matthies [87] used a hand-defined cost function based on the distance of the robot to the goal, nearby obstacles, and unknown space to approximate the costs of abstract actions in a grid environment. Other approaches have used properties of the primitive plan space to approximate the costs of abstract actions; for example, one approach used wavelets to approximate the costs of abstract actions based on terrain smoothness in a grid cell, computed from a known, primitive terrain map [67]. More recently, another approach used deep learning to approximate the costs and traversability of abstract actions, conditioned on local primitive height map data [44]. While these approaches have been demonstrated to increase planning speeds for some problems, they do not incorporate uncertainty into abstract actions, and their abstract plans ultimately serve as inadmissible heuristics for the planning problem in the primitive space.

Another line of work has focused on generating hierarchical abstractions that maintain planning guarantees. Marthi et al. (2007) and Marthi et al. (2009) [55, 56] proposed an angelic semantics for abstract actions. By assuming that uncertainties in the costs of abstract actions could be resolved in favor of the planner, the authors defined a hierarchical planning approach that used bounds on the costs of abstract actions to compute hierarchically optimal plans. This approach was extended by Vega-Brown and Roy [90] to guarantee the generation of primitively optimal plans, assuming admissible bounds on the costs of abstract actions. Specifically, Vega-Brown and Roy [90] defined a metric decomposition of the environment into convex regions and calculated analytic upper and lower bounds on the costs of primitive plans passing through the convex regions using Hausdorff and Euclidean distances, assuming a path length cost function. While this technique effectively solved planning problems that could be decomposed into a small number of convex regions and used a path length cost function, it is not clear that the algorithm would scale to noisy, real-world environments where sensor noise may lead to an inability to generate a finite set of convex regions over which to plan, and where the path length cost function

may not be sufficient to encode desired robot behavior.

## 2.6 Informed Planning

In Sections 2.3, 2.4 and 2.5, we discussed vanilla implementations of common planning paradigms. However, a wide body of literature has focused on improving planning paradigms for use in task-specific planning scenarios. In this section, we discuss modifications to common planning algorithms that use task or domain knowledge to improve planning outcomes. We refer to these planners as *informed* planners.

### 2.6.1 Informed Model Construction

The goal of informed modeling is to generate models of the environment that improve model representativeness or tractability. In the following section, we discuss informed modeling for discretized configuration space models and sampling-based models. We also briefly discuss informed cost functions, which can be used to approximate the costs of discrete actions, if they are unknown.

#### Informed Configuration Space Discretization

To mitigate the challenges of representing discrete configuration spaces using a constant resolution discretization or a hierarchy of constant resolution discretizations, a number of methods have been proposed that non-uniformly decompose the configuration space based on properties of the space itself. For example, some approaches increase the resolution of the state decomposition near the surfaces of objects [10]. Other methods increase the resolution of the decomposition near the agent [8, 64].

#### Informed Sampling for Sampling-based Models

A variety of approaches have used informed sampling distributions $\Phi(x; \Gamma)$ to increase the quality and efficiency of sampling-based models. Specifically, these approaches use sampling to increase the probability of quickly generating representative, tractable

graphs of the environment over which to search. In this section, we discuss a number of informed sampling strategies.

One class of informed sampling distributions focuses on using obstacle information to inform sampling. An early example of this paradigm was bridge sampling, which was originally proposed to improve the efficiency of Markov-Chain Monte Carlo sampling methods [9], and was later adapted to the robot motion planning domain [34]. In bridge sampling, pairs of samples were initially drawn uniformly from the environment. Then, if both samples were in collision, the midpoint between the samples was also considered and added to the graph, if unoccupied. This method has been particularly successful in environments with narrow passageways, where samples in collision with passage walls generate non-colliding states inside the passageway, increasing the probability of generating a valid path through the passageway. Other approaches have similarly relied on using occupied samples to sample more efficiently in unoccupied areas of the state space. The Gaussian sampler biased sampling away from large open areas of the state space by generating two nearby samples per iteration and only adding a sample to the graph if one and only one of the samples was non-colliding [11]. Obstacle-based sampling generated samples that traced obstacle boundaries by sampling in obstacle regions, then traversing in a direction until a valid region was encountered [5, 95]. Other approaches biased sampling away from obstacles using a proximity look-up table based on previous collision checks [45].

Other works used geometry-based environmental decompositions to inform intelligent sampling. A medial-axis-based approach used the Delaunay triangulation of a Voronoi decomposition of the state space to approximate the medial axis of the space, then explicitly sampled along the medial axis to increase the probability of generating a connected graph of the environment [28]. Another approach decomposed the continuous state space into discrete cells, then sampled non-uniformly from the cells based on local geometric characteristics, like narrow passageways [89].

Another line of work generated adaptive, explorative sampling strategies. Burns and Brock (2005a) and Burns and Brock (2005b) [12, 13] generated an approximate model of the configuration space online, and biased sampling to regions of the envi-

ronment that minimized the variance of the environmental model; this encouraged the planner to explore novel regions. Siméon et al. [81] used visibility domains to sparsify sampling while maintaining or improving graph connectivity; Yershova et al. [96] introduced Dynamic-Domain RRTs, which adapt the visibility approach to avoid over-exploration of Voronoi regions containing obstacles. Shkolnik and Tedrake [77] introduced the BallTree method, which encouraged exploration by rejecting samples that were near the existing tree.

Other planners calculated approximations of sample quality to bias sampling towards regions of the configuration space that were likely to lead to low cost solutions. Urmson and Simmons [88] used a sample quality metric, based on cost-to-come and cost-to-go, to bias sampling to Voronoi regions that were likely to contain low-cost solutions to the planning problem. Zucker et al. [100] used the REINFORCE Algorithm to identify discretized regions of the environment that were often used in optimal navigation strategies, then biased sampling towards these regions.

Yet other techniques used properties of solution paths to improve solution quality for anytime techniques. Islam et al. [37] used the triangle inequality to directly improve upon existing solutions, once found, and biased sampling towards regions near solutions, which encouraged the refinement of existing solutions. Gammell et al. [22] used constraints on geometric path length to restrict sampling to areas of the environment that could improve an existing solution, once one was found.

Recently, a number of approaches have applied learning using geometric information to the intelligent sampling problem for sampling-based motion planning. Ichter et al. [35] used a conditional variational autoencoder to learn sampling strategies for sampling-based motion planners from optimal trajectories, conditioned on available occupancy information. The approach placed samples in narrow passageways and along likely solution paths, and increased the probability of finding valid trajectories while maintaining or improving upon solution cost, as compared to a sampling-based motion planner using a uniform sampling distribution. Ichter et al. [36] and Molina et al. [59] used learning to effectively build graphs for sampling-based motion planning in critical map regions, defined using the graph-theoretic measure betweenness

centrality; Molina et al. [59] used a convolutional neural network to classify regions of the environment as critical or non-critical, then directly sampled from critical regions; Ichter et al. [36] used a neural network to classify samples as critical or non-critical, then used a global connection strategy for critical samples. Wang et al. [91] used a convolutional neural network to learn a discretized probability distribution from optimal trajectories, which increased the probability of sampling in regions that were likely to occur on low-cost trajectories. Zhang et al. [97] used the REINFORCE algorithm to learn a rejection sampler conditioned on the locations of existing samples, and demonstrated the ability to decrease planning time by decreasing the number of expensive operations taken in the planning loop, like collision checks and adding nodes to the tree. While each of these approaches demonstrated improved planning performance, they all relied on geometric information to inform graph construction for sampling-based motion planning.

### Informed Edge Addition for Sampling-based Models

A variety of approaches have been proposed to modify edge addition strategies for sampling-based motion planning; these approaches focused on efficiently building graphs that were likely to contain low-cost solutions to the planning problem. Jaillet et al. [38] biased tree growth towards minimum-work paths, defined using a continuous cost function, using a Metropolis transition acceptance step; this decreased the likelihood of adding high-cost nodes that were unlikely to be part of low-cost plans to the tree. More recently, Janson et al. [39] introduced the Fast Marching Trees (FMT*) algorithm, which used a marching method to order batch tree edge addition based on cost-to-come; this ordering biased tree growth to edges that were likely to contain low-cost paths. Finally, Gammell et al. (2015) and Gammell et al. (2020) [23, 24] introduced Batch Informed Trees (BIT*), which constrained tree growth to edges in ellipses that defined low-cost planning solutions, extending informed tree growth to the anytime setting.

### Informed Cost Functions

Finally, we consider informed cost functions, which use properties of the environment to approximate the costs of discrete actions, when these costs are not known. This approach has been particularly useful for navigation in outdoor environments with complex terrain types. One approach learned properties of environmental traversals from overhead images, but relied on a hand-coded function to transform the environmental properties into a cost function for a discrete planner [78]. Later, the maximum entropy inverse reinforcement learning approach generated a parameterized cost function for navigation by maximizing the probability of recovering expert behavior using the cost function [99]; for example, the approach placed high cost in regions of the environment that experts traversed infrequently, while placing low cost in regions of the environment that experts traversed frequently. However, this method and other similar approaches required the optimizer to consider the relative costs of all trajectories in an environment, complicating their application to large domains, where it can be infeasible to consider the costs of all possible plans simultaneously [79, 80]. More recently, deep learning approaches have been proposed to learn cost functions for the navigation task. One approach extended maximum entropy inverse reinforcement learning to deep neural networks [93, 94], and learned cost functions based on occupancy maps built online.

### Challenges of Informed Model Construction

Despite their ubiquity in the literature, few of the above informed planning strategies are regularly incorporated into real-world planners. One of the main limitations of the presented approaches is their reliance on high-quality geometric information, which is often unavailable due to sensor range and quality, or is insufficient to inform intelligent planning decisions in complex environments. In Section 2.7, we motivate the use of multimodal sensor information to inform robot navigation.

### 2.6.2 Informed Search

The goal of informed search is to define search algorithms that are more efficient or find lower cost plans than traditional search algorithms. In the following section, we discuss informed heuristics for robot motion planning.

#### Informed Heuristics

In this section, we review a number of heuristics for informed discrete search. As discussed in Section 2.4.1, heuristics for discrete planning are used to direct search to areas of the environment that are likely to lead to low cost solutions. However, accurate heuristics can be expensive to compute, especially in structured environments, where common heuristics like Euclidean distance produce inaccurate cost estimates. Two example failure cases of the Euclidean distance heuristic in structured environments are shown in Figure 2-1. In both examples, the structure of the office building indicates that the robot should follow hallways to traverse between offices or to exit buildings. However, the overly optimistic Euclidean distance heuristic encourages the robots to enter dead-end rooms in an attempt to greedily minimize the distance to the goal.

To avoid these inefficiencies, a number of approaches for online multi-query planning precompute a limited number of traversal costs in known environments, then used properties of the known traversals to approximate heuristics for novel queries. The A*, Landmark and Triangle Inequality (ALT) and Landmark Pathfinding between Intersections (LPI) algorithms used landmarks to sparsely precompute traversal costs in an environment, then used landmark vertices as waypoints when calculating heuristics for online planning [26, 27]. These approaches were extended to the robotics domain by Murphy and Newman [62], which used ALT to approximate traversal costs in complex, outdoor terrains. While these approaches can be very efficient in multi-query settings for deterministic environments, it can be expensive to precompute traversal costs in an environment, even for a limited number of landmarks. Therefore, these methods are most useful for problems where the computational cost of calculating


Figure 2-1: **Two examples where the Euclidean distance heuristic fails in structured environments.** Each image depicts a real-world university floorplan, where occupied space is shown in white and unoccupied space is shown in black. The start and goal are depicted using green and red circles, respectively. The optimal trajectory calculated using the A* algorithm is shown in blue; the Euclidean distance heuristic is shown in green. In both examples, the structure of the environment indicates that the robot should travel through hallways to reach the goal, but the Euclidean distance heuristic assumes the robot can take shortcuts through walls, and severely underestimates the true costs of the optimal plans. We propose using non-geometric information to enable the robot to plan efficiently in structured environments.

landmark properties can be amortized over many queries.

### Challenges of Informed Search

While clever heuristics can improve planning results for discrete planners, these approaches are still fundamentally limited by their reliance on high-resolution data to generate accurate heuristics. In Section 2.7, we discuss the use of non-geometric data to inform intelligent navigation strategies in the absence of complete geometric information.

### 2.6.3 Informed Hierarchical Planning

In this section, we review a number of state-of-the-art hierarchical planners that used properties of the navigation task to inform the abstract representation of the action space. A number of abstractions have been proposed to complete the navigation task by cleverly decomposing the environment into representations that make inference about abstract plan properties easy. Stein et al. (2018) and Stein et al. (2020) [82, 83] decomposed navigation in unknown environments into traversals between frontiers between known and unknown space in an occupancy map, allowing for the efficient factorization and computation of the Bellman Equation despite the high dimensionality of partially-observed environments. Vega-Brown and Roy [90] defined an abstraction for navigation by decomposing the environment into a finite set of regions through which traversals occured, and used convexity assumptions to analytically compute upper and lower bounds for the costs of traversals through the regions. Larsson et al. (2019) and Larsson et al. (2020) [48, 49] defined an abstraction for navigation by compressing a known probabilistic occupancy map using information theoretic measures and a hand-tuned computational budget, and generated low cost plans while considering a smaller portion of the state space than traditional methods. Other approaches used deep learning to define an abstract state space and its properties; Klamt and Behnke [44] used a convolutional neural network (CNN) to learn a grid-based abstract representation of an environment from primitive geometric height map data, then learned properties of the representation using a fully connected network (FCN).

#### Challenges of Informed Hierarchical Planning

While each of these approaches generated good plans when the abstraction and its properties could be easily computed from primitive geometric information, it is not clear how to extend these representations to environments where geometric features are expensive to compute and maintain or insufficient to encode intelligent navigation strategies. In Section 2.7, we motivate the use of non-geometric information to inform

hierarchical planning.

## 2.7 Non-Geometric Information to Inform Planning

In this section, we motivate the use of non-geometric information, namely semantic information (i.e., explicit non-geometric information) and perceptual abstractions (i.e., implicit non-geometric information), to inform planning. We also motivate the use of *multimodal* information, or the simultaneous use of multiple types of information, to inform planning. We consider biologically plausible navigation strategies, and review the use of non-geometric information for planning.

### 2.7.1 Biologically Inspired Planning

Roboticists can gain significant insight into successful motion planning by considering biological planning strategies. Overwhelmingly, it has been shown that biological agents reason about navigation decisions using non-geometric data. For example, it has been shown that rats use head direction cells to determine absolute orientation, place cells to identify specific locations, hexagonal networks of grid cells to specify relative translations, and border cells to identify edges of closed environments [18]. It been shown that humans can detect navigable regions in novel environments with no increase in cognitive load [66], and that the human hippocampus uses both Euclidean distance heuristics and turn-by-turn navigation information when planning [16]. These anecdotes provide significant evidence that humans and other mammals use multimodal strategies when making planning decisions.

Beyond human physical sensory input and data storage, it has been demonstrated that humans do not require dense geometric knowledge to make planning decisions. In one study of the navigation behaviors of novice and expert taxi drivers, drivers were asked to complete geometric map-based tests, like map drawing and place location on maps, and then were tasked with completing taxi routes. While the expert group demonstrated improved navigation performance when completing taxi routes, they showed no improvement over the novice group in the geometric tasks, indicating that

human navigation behavior is not dependent on high-fidelity geometric mapping. Additionally, the study notes that both novices and experts engaged in on-the-fly route improvement, but that experts where better at identifying locations from single images [15]. These findings further demonstrate that human navigation behavior is far more complex than simple geometric path planning; however, geometric path planning has dominated the robotic planning literature. We hypothesize that multimodal information, if correctly incorporated into planning decisions, could improve robot planning outcomes.

### 2.7.2 State-of-the-art Multimodal Robot Navigation

While biology indicates that navigating using non-geometric cues could improve robot planning, it is not obvious how to represent multimodal information for planning. State-of-the-art approaches use domain-specific, hand-crafted models to incorporate non-geometric information into intelligent planning decisions. For example, Pronobis et al. [72] presented the Deep Spatial Affordance Hierarchy (DASH), a four-level probabilistic topological mapping scheme designed to specifically support complex robot reasoning and planning using multimodal data. At the lowest level of the hierarchy, DASH represented the environment using local body-centered occupancy maps. At the highest level of the hierarchy, the model encoded probabilistic semantic knowledge, such as probabilistic object detections and probabilistic room categorizations. While the model was successful for some tasks, including semantic place categorization, novelty detection, occupancy map generation, and occupancy map completion in an office environment [72, 98], it relied heavily on hand-crafted interactions between topological levels, and may be challenging to generalize to larger environments with more complex inter-element and inter-layer interactions. The cognitive science community introduced SemaFORR [19], an affordance-based decision-making system based on learned spatial affordances and commonsense reasoning. However, SemaFORR relied on three hand-crafted affordance categories and a hand-crafted hierarchical decision-making system, and it is not clear how the approach could be extended to include a more diverse set of affordances.

While these approaches successfully demonstrated navigation tasks in a small number of simple environments, significant engineering effort would be required to apply the approaches to larger or more complex environments. First, they require an expert to define how data from different modalities should interact with each other to inform planning. Second, because they use novel data structures to explicitly represent task-specific hierarchical environmental interactions, they cannot be used in conjunction with existing planning algorithms, and instead rely on model-specific planning techniques. This limits their applicability to different agents navigating in diverse environments.

### 2.7.3 Metrically Aligned Non-Geometric Information for Robot Navigation

As stated above, one of the key challenges for using multimodal data for planning is determining how different modalities of data should be synthesized to generate planning decisions. Existing approaches rely on expert knowledge to explicitly encode relationships between non-geometric information and intelligent navigation behaviors. However, planning optimally using geometric information alone is a well understood problem. In this thesis, we realize that we can use geometric information to generate optimal behaviors in an environment while simultaneously collecting non-geometric information. After navigation terminates, we can correlate non-geometric information to optimal geometric navigation behaviors. However, in order to correlate optimal geometric trajectories and non-geometric information, we need to define non-geometric information relative to a metric coordinate frame. In this section, we review two types of non-geometric information – object-level maps and dense perceptual maps – that can be metrically aligned.

#### Object-Level Maps

One approach to storing non-geometric information in a metric coordinate frame is object level mapping; this approach is particularly useful when objects in an environ-

ment inform intelligent navigation behaviors. For example, consider the scenario in Figure 2-2a, where a robot (green circle) attempts to navigate in a partially known environment to reach a goal in an adjacent room (red circle). Doors are shown in yellow, windows in blue, and walls in dark gray. The robot has mapped the white portions of the world; all other areas are unmapped (light gray overlay). Without object-level information, the robot will greedily attempt to navigate through the wall to the goal (red dashed line). However, if the robot is equipped with an object-based reasoning system and sees a door, it can recognize that rooms are likely connected via hallways, and that doors are ways to exit rooms to get to hallways (green dashed line), independent of the partially mapped dense geometry of the room. Additionally, consider a robot attempting to exit a building (Figure 2-2)b; if the robot has access to an object-based reasoning system and sees a door with an exit sign, it can exit the building through the door (green dashed line), instead of greedily exploring a room in an attempt to minimize the geometric distance to the goal (red dashed line). However, to enable this behavior, the robot must have knowledge of the locations of the doors in the environment.

Fortunately, a number of object-level mapping systems have been proposed. SLAM++ initially demonstrated online 6-DOF object localization and mapping using known CAD models of objects [76]; since then, a variety of CAD-based object-level SLAM systems have been proposed [30, 31]. For applications where object CAD models are not available, or where CAD-level detail is not required for successful task completion, such as obstacle avoidance [65], alternative object representations have been proposed. QuadricSLAM generated object-level maps by using bounding box detections to represent objects as quadrics using the dual quadric form [63]; ROSHAN extended this representation by introducing a texture plane and semantic shape prior to improve quadric initialization for online planning [65].

By representing objects in a compact way, object-level maps have enabled intelligent object-based motion planning. For example, Sünderhauf [85] used graph convolutional networks to learn navigation policies for object search from graphs built by object-level mapping systems. Baldwin and Newman [6] used a non-parametric model

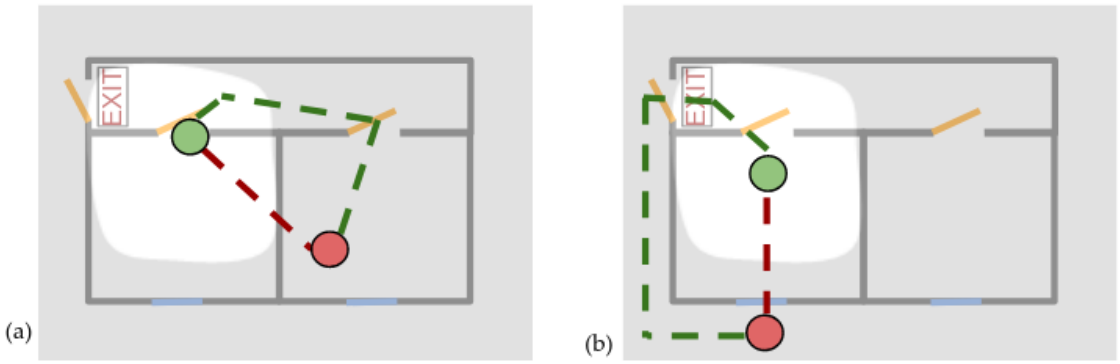


Figure 2-2: **Two examples where object-level information can inform navigation in unknown environments.** Consider a robot (green circle) attempting to navigate to a goal (red circle) in a partially mapped environment. Doors are shown in yellow, windows in blue, and walls in dark grey. The robot's observed map is shown by the white region; the grey overlay indicates that the area of the environment has not been mapped. (a) The robot attempts to navigate to a goal in an adjacent room. Without object-level information, the robot will greedily attempt to navigate through the wall to the goal (red dashed line). However, if the robot is equipped with an object-based reasoning system and sees a door, it can recognize that rooms are likely connected via hallways, and that doors are ways to exit rooms to get to hallways (green dashed line), independent of the partially mapped dense geometry of the room. (b) The robot attempts to navigate to a goal that is outdoors. If the robot has access to an object-based reasoning pipeline and sees a door with an exit sign, it can exit the building through the door (green dashed line), instead of greedily exploring a room in an attempt to minimize the geometric distance to the goal (red dashed line).

to learn a sampling distribution relative to objects with known positions, including traffic circles and T-intersections. In Chapter 3, we present Learned Sampling Distributions, a sampling-based motion planning approach that uses optimal trajectories to learn an informed sampling distribution and cost function from partially-observed geometric and object-level maps to improve robot navigation outcomes in unknown environments.

### Dense Perceptual Maps

Another approach to storing non-geometric information in a metric coordinate frame is dense perceptual mapping, a term we use to describe dense representations of implicit non-geometric information (e.g., semantic segmentation, metrically-correlated

overhead images); this approach is particularly useful when non-object elements of the environment inform intelligent navigation behaviors. For example, consider a robot navigating in an outdoor environment with a forest and a field; we would like for the robot to infer that paths through the forest are likely to be higher cost than paths through the field. Some approaches use semantically segmented images to approximate cost functions for discrete planning from overhead images; Murphy and Newman (2010) and Murphy and Newman (2012) [61, 62] learned a Gaussian Process to segment an overhead image into terrain classes, then learned spatially-informed cost functions for each terrain class using additional spatially-informed Gaussian Processes. Wigness et al. [92] used one-hot semantically segmented maps as inputs to a neural network that optimized a maximum entropy inverse reinforcement learning objective to learn a cost function for navigation in an outdoor environment using a discrete planner. Everett et al. [20] used semantically segmented overhead images to learn a cost-to-go heuristic for the last mile delivery task.

While existing approaches to planning using dense perceptual maps have been successful for short-horizon planning tasks, these approaches were devised using discretized state models and sampling-based models, which can be inefficient when applied to long length scale navigation. In Chapter 5, we present Perceptually Informed Abstractions, a preliminary hierarchical planning approach that learns validity and cost functions for abstract actions using overhead image data to increase planning efficiency in long length scale environments.

# Chapter 3

# Learned Sampling Distributions

*The research described in this chapter was performed in conjunction with Katherine Liu.*

In this chapter, we present Learned Sampling Distributions, a sampling-based motion planning (SBMP) algorithm that enables intelligent, efficient robot planning in unknown environments. We discuss the challenges posed by introducing unknown space into the formulation for discrete planning over graphs in Equation 2.5, and motivate the use of hybrid geometric and semantic information to improve sampling efficiency in these environments. We describe a method for learning sampling distributions from example optimal trajectories in known environments, and formalize the optimization over sampling distributions as the learning of network weights in a convolutional neural network.

## 3.1 Optimal Planning in Unknown Environments

In Section 2.3.2, we defined the optimal robot motion planning problem over graphs $G$ as a minimization of a cost function over all possible robot trajectories $\boldsymbol{C}(\mathcal{T})$, subject to validity constraints; specifically, we required each vertex $v$ and edge $e$ to be valid, $f_v(v) = 1 \ \forall v \in V$ and $f_e(e) = 1 \ \forall e \in E$. However, in environments where dense geometry is not known, the validity of each vertex and edge cannot be checked directly, and additional information is needed to generate graphs that are likely to

contain valid, low cost trajectories to solve the optimization in Equation 2.5.

In Section 2.3.3, we motivated the use of sampling-based methods for efficient graph construction. As discussed in the section, there are three steps in the sampling-based motion planning process:

1. **Sample Vertices:** Randomly draw states $x$, defined over the configuration space $\mathcal{X}$, to represent as candidate vertices $v$ from a sampling distribution, $\Phi(x; \Gamma)$ where $\Gamma$ are optional additional inputs to the distribution. Add valid vertices to the planning graph.

2. **Add Edges:** Connect vertices in the planning graph using a deterministic edge addition strategy. Label the edges according to a known cost function $\boldsymbol{C}(\mathcal{T})$[1].

3. **Search Graph:** Use a standard graph search algorithm to find the shortest path in the graph (for example, Hart et al. [29], Dijkstra et al. [17]).

Intuitively, a good choice of sampling distribution is a distribution which generates graphs that are more likely to contain solutions to the optimization problem in Equation 2.5. In fully observed environments, a variety of techniques have been developed to sample efficiently in the presence of environmental structure by using fully-known occupancy maps, $M_g^*$, and the geometric location of the goal, $x_g$, as conditioning parameters for the sampling distribution, $\Gamma = \{M_g^*, x_g\}$ [96, 13, 7]. However, these methods cannot be applied in unknown environments, where $M_g^*$ is either unavailable or incomplete.

Additionally, in order to generate good paths in unknown environments, an uncertainty-aware cost function must be used to calculate edge costs. In fully-known environments, cost functions such as path length are sufficient to ensure good planning performance, as all edges that are added to the graph necessarily indicate valid traversals of the environment due to collision checking. However, in unknown environments, where collision checking cannot be performed directly, graph edges do not necessarily indicate valid traversals in the environment. Because graph edges can be invalid,

[1]We note that the trajectory cost function can be used to determine the costs of edges in the graph, as each edge is a valid trajectory.

minimum length paths in unknown environments are often overly optimistic about the traversability of unknown space in the environment, and lead to unsuccessful plans once additional occupancy information is collected. To avoid over-optimistic planning behavior, we propose an uncertainty-aware cost function that increases the probability of selecting a valid trajectory in unknown environments.

## 3.2 Multimodal Information for Navigation

To sample intelligently in unknown environments, we propose defining a sampling distribution $\Phi(x; \Gamma)$ parameterized by geometric and semantic information collected online. While complete geometric information is not available in unknown environments, a variety of inexpensive sensors can provide an agent with a rich set of geometric and non-geometric information in real-time. Off-the-shelf depth cameras, like the Intel Realsense Depth Camera D435i, can provide accurate local depth information to about 5m and RGB images at visual range [2]. Using open source online occupancy mapping schemes [32] and more recent online metrically-correlated object-level mapping systems [63, 65], we can generate partially-observed, metrically-correlated geometric and object-level maps that contain important navigational cues for navigating in unknown environments in real-time; we call the set of geometric and object-level maps a hybrid map, $\mathcal{M} = \{M_g, M_s\}$. Formally, we parameterize the informed sampling distribution by $\mathcal{M}$, as well as additional task-level contextual information, $\mathcal{C}$, $\Gamma = \{\mathcal{M}, \mathcal{C}\}$. In practice, $\mathcal{C}$ contains the geometric location of the goal relative to the robot, and a boolean indicator that indicates whether the robot's goal is located indoors or outdoors. We generate graphs using the resulting sampling distribution,

$$\Phi(x; \{\mathcal{M}, \mathcal{C}\}), \tag{3.1}$$

to generate candidate solutions to the optimization in Equation 2.5.

Building a graph using an informed sampling distribution is not sufficient to generate high-quality solutions in unknown environments; the robot must also define a

cost function over trajectories that prioritizes trajectories that are likely to lead to the goal. Unfortunately, common trajectory cost functions, like Euclidean distance, are insufficient to generate informed trajectories in unknown environments; in the limit of a graph with infinite samples, a Euclidean distance cost function reverts to greedy shortest-path behavior which is likely to lead to invalid plans once the underlying geometry of the environment is observed. Instead, we propose using the sampling distribution not only to sample the planning graph, but also as a cost function over graph edges:

$$\boldsymbol{C}(\mathcal{T}) \propto -\log(\Phi(\mathcal{T}; \{\mathcal{M}, \mathcal{C}\}). \tag{3.2}$$

## 3.3 Learned Sampling Distributions

In general, determining the exact form of $\Phi(x; \{\mathcal{M}, \mathcal{C}\})$ is a high-dimensional, non-convex optimization problem that is no easier to solve than the original planning problem. However, existing work from the planning under uncertainty community indicates that local environmental cues can inform productive navigation behaviors in unknown environments [82]. For this reason, we choose to approximate the probability of sampling an optimal trajectory $\mathcal{T}^*$ with a mapping $\phi$ that depends on the hybrid map and the contextual information,

$$\Phi(\mathcal{T}^*; \mathcal{M}_i, \mathcal{C}_i) \approx \phi(\mathcal{T}^*, \alpha; \mathcal{M}_i, \mathcal{C}_i), \tag{3.3}$$

where $\alpha$ indicates the parameters of the predictive sampling distribution model.

While generating a dataset of example optimal probability distributions is challenging and often intractable, generating a dataset of optimal trajectories in example environments is a more tractable endeavor. We propose collecting a dataset of optimal trajectories generated offline using an optimal path planning algorithm in known maps. Formally, we collect a dataset $\mathcal{D}$ of $q$ example optimal trajectories $\mathcal{T}^*$ and partial hybrid maps $\mathcal{M}$ and context $\mathcal{C}$,

$$\mathcal{D} = \{(\mathcal{T}_0^*, \mathcal{M}_0, \mathcal{C}_0), (\mathcal{T}_1^*, \mathcal{M}_1, \mathcal{C}_1), ..., (\mathcal{T}_q^*, \mathcal{M}_q, \mathcal{C}_q)\}. \tag{3.4}$$

We propose learning a sampling distribution by maximizing the probability that optimal trajectory $\mathcal{T}^*$ will be sampled, parameterized by the available map and context information:

$$\arg\max_{\alpha} \prod_{i}^{q} \Phi(\mathcal{T}_i^*, \alpha; \mathcal{M}_i, \mathcal{C}_i). \tag{3.5}$$

For numerical stability, we convert the maximization problem into a minimization of the negative log-likelihood:

$$\arg\min_{\alpha} \sum_{i}^{q} -\log\Phi(\mathcal{T}_i^*, \alpha; \mathcal{M}_i, \mathcal{C}_i). \tag{3.6}$$

However, the joint distribution over $V$ and $E$ is still challenging to optimize, as the space of all possible sampled graphs is exponentially large. Instead, we approximate the optimization over sampled graphs as an optimization over graph vertices only:

$$\arg\min_{\alpha} \sum_{i}^{q} -\log\Phi(V_i, \alpha; \mathcal{M}_i, \mathcal{C}_i). \tag{3.7}$$

However, this joint distribution is still expensive to compute. To maintain computational tractability, we make the assumption that individual vertices in a trajectory are conditionally independent:

$$\arg\min_{\alpha} \sum_{i}^{q} -\log(\prod_{j=0}^{|\mathcal{T}_i|} \Phi(v_{ij}, \alpha; \mathcal{M}_i, \mathcal{C}_i)). \tag{3.8}$$

Substituting our neural network approximation for the true probability distribution, we recover our final optimization:

$$\arg\min_{\alpha} \sum_{i}^{q} (\sum_{j=0}^{|\mathcal{T}_i|} -\log\phi(v_{ij}, \alpha; \mathcal{M}_i, \mathcal{C}_i)). \tag{3.9}$$

In practice, we optimize a scaled version of Equation 3.9 for numerical stability.

## 3.4 Neural Network Model Structure and Optimization

While a number of optimization schemes could be used to optimize Equation 3.9, in this work, we choose to implement $\phi$ using a Convolutional Neural Network (CNN), due to its known success in representing 2D spatial relationships. Specifically, we define $\phi$ as a neural network with three distinct parts: $\phi^{enc}$, which projects the input hybrid map $\mathcal{M}$ into a low-dimensional latent space, $\phi^{dec}$, which lifts the low-dimensional latent space into the probability space, and outputs a task-independent learned sampling distribution, and $\phi^{s}$, a task-specific distribution modifier that takes the learned latent space and context parameters $\mathcal{C}$ as input and outputs a task-specific mask, which is multiplied with the task-independent distribution to produce an unnormalized task-specific distribution. Finally, the unnormalized distribution is normalized to generate the final learned sampling distribution, $\phi$. The complete neural network structure is shown in Figure 3-1.

We represent $\mathcal{M} = [M_g; M_s]$ as a multi-channel map, where each channel contains a single mode of information, such that $M_g, M_s \in \mathbb{R}^{k \times k}$, and $\mathcal{M} \in \mathbb{R}^{k \times k \times 2}$, where $k \in \mathbb{Z}_{>0}$. We generate the low-dimensional latent space, $l$, by applying the encoding function $\phi^{enc}$ to $\mathcal{M}$,

$$l = \phi^{enc}(x_d; \mathcal{M}, \alpha_{enc}), \tag{3.10}$$

where $x_d \in \mathcal{X}_d$, $\mathcal{X}_d$ is a discretized form of $\mathcal{X}$, and $\alpha_{enc}$ denotes the network weights for the encoding network.

Next, we apply the decoding function to the learned latent layer, and recover a task-agnostic probability distribution, which encodes areas of the environment that are likely to be on optimal paths. For example, this distribution qualitatively places high probability in areas like hallways and doorways, which are often found on optimal traversals through office buildings.

$$\phi^{gen}(x_d; \mathcal{M}, \alpha_{\{enc,dec\}}) = \phi^{dec}(l; \alpha_{dec}), \tag{3.11}$$

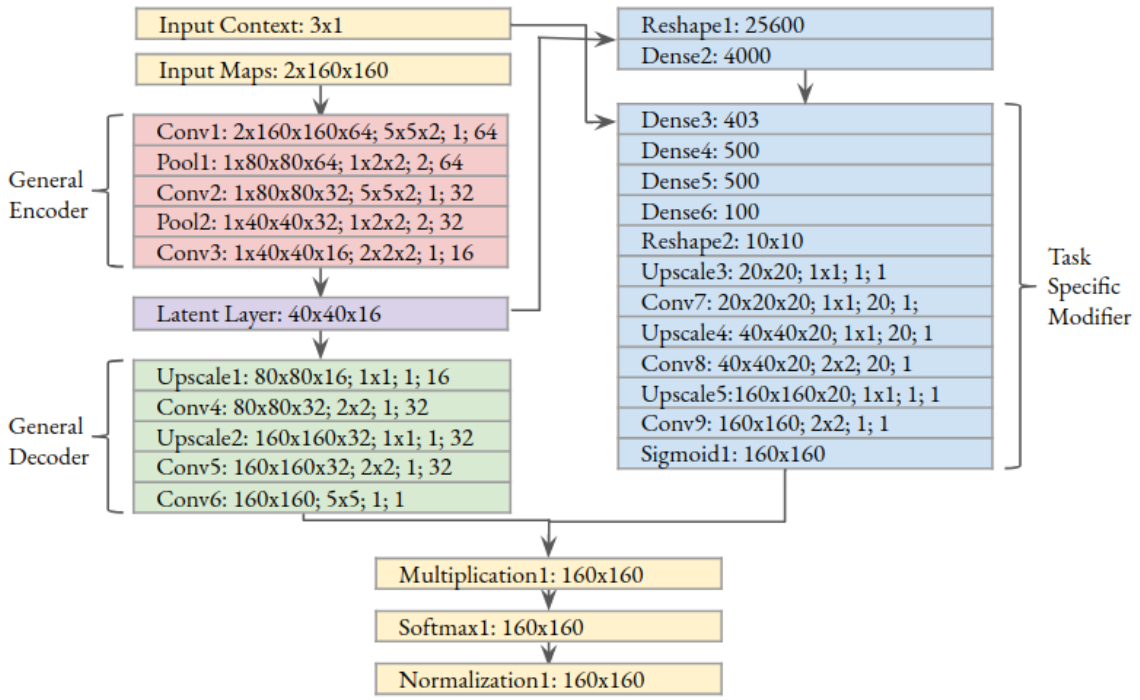


Figure 3-1: **Neural network used to generate learned sampling distributions.** The network uses geometric and semantic information to learn general and task specific sampling distributions that empower a robot to reason about navigational modes in unknown space. Layer data indicates, in order, layer size, convolution size, stride size, and number of filters. Dropouts are not pictured but used for regularization. The context information is a 3-vector with the first two values proportional the relative location of the goal, and the third value is a scaled indicator variable indicating whether the goal is inside or outside.

where $\alpha_{dec}$ denotes the network weights for the decoding network.

However, general navigation strategies are not sufficient to inform optimal task-based planning; for instance, imagine attempting to navigate between two offices in an office building. While hallways generally indicate a fast method of traversal through a building, this information is bidirectional, and does not capture the directionality necessary to enable efficient planning from one specific office in a building to a second specific office in a building. Similarly, the utility of exit signs depends on whether or not a robot needs to exit a building to reach its goal. To enable task-specific planning, we represent task-specific information in a context parameter, $\mathcal{C} = [x_g^{rel}, I_{out}]$, where $x_g^{rel}$ is the location of the goal in the robot frame, and $I_{out}$ is a discrete indicator that indicates whether the goal is indoors or outdoors, such that $\mathcal{C} \in \mathbb{R}^{d+1}$. We account

for task-specific behaviors by learning a task-specific distribution modifier $\phi^s$, given the latent layer $l$ and parameterized by context $\mathcal{C}$,

$$\phi^s(x_d; \mathcal{M}, \mathcal{C}, \alpha_{\{enc,s\}}) = \phi^s(l; \mathcal{C}, \alpha_s), \tag{3.12}$$

where $\alpha_s$ denotes the network weights for the task-specific distribution modifier network.

To generate an unnormalized task-specific distribution $\phi^u$, we multiply the general distribution $\phi^{gen}$ with the task-specific distribution $\phi^s$:

$$\phi^u(x_d; \mathcal{M}, \mathcal{C}, \alpha_{\{enc,dec,s\}}) = \phi^{gen}(x_d; \mathcal{M}, \alpha_{\{enc,dec\}}) \times \phi^s(x_d; \mathcal{M}, \mathcal{C}, \alpha_{\{enc,s\}}). \tag{3.13}$$

Finally, to generate a proper probability distribution, we apply a softmax layer to the neural network in Equation 3.13:

$$\phi(x_d; \mathcal{M}, \mathcal{C}, \alpha_{\{enc,dec,s\}}) = \text{softmax}(\phi^u(x_d; \mathcal{M}, \mathcal{C}, \alpha_{\{enc,dec,s\}})). \tag{3.14}$$

In practice, we employ a multi-stage optimization process to optimize $\phi$. First, we optimize $\alpha_{enc}$ and $\alpha_{dec}$ while holding $\alpha_s$ constant to learn a task-independent sampling distribution. Then, we freeze $\alpha_{enc}$ and $\alpha_{dec}$ and optimize $\alpha_s$ to learn a task-specific distribution modifier. The network optimizes Equation 3.9 via back-propagation and stochastic gradient descent (SGD) with a learning rate of 0.0005 and mini-batches of 500 data points. We provide examples qualitative network outputs in Figures 3-2 and 3-3.

## 3.5 Online Planning in Unknown Environments

To use the learned distribution online, we first optimize Equation 3.9 offline by applying SGD to the structure in Equation 3.14. During each timestep of online use, we generate robot-centered incomplete local geometric and object-level maps of the environment using a sliding window, then run feed-forward prediction on our pre-

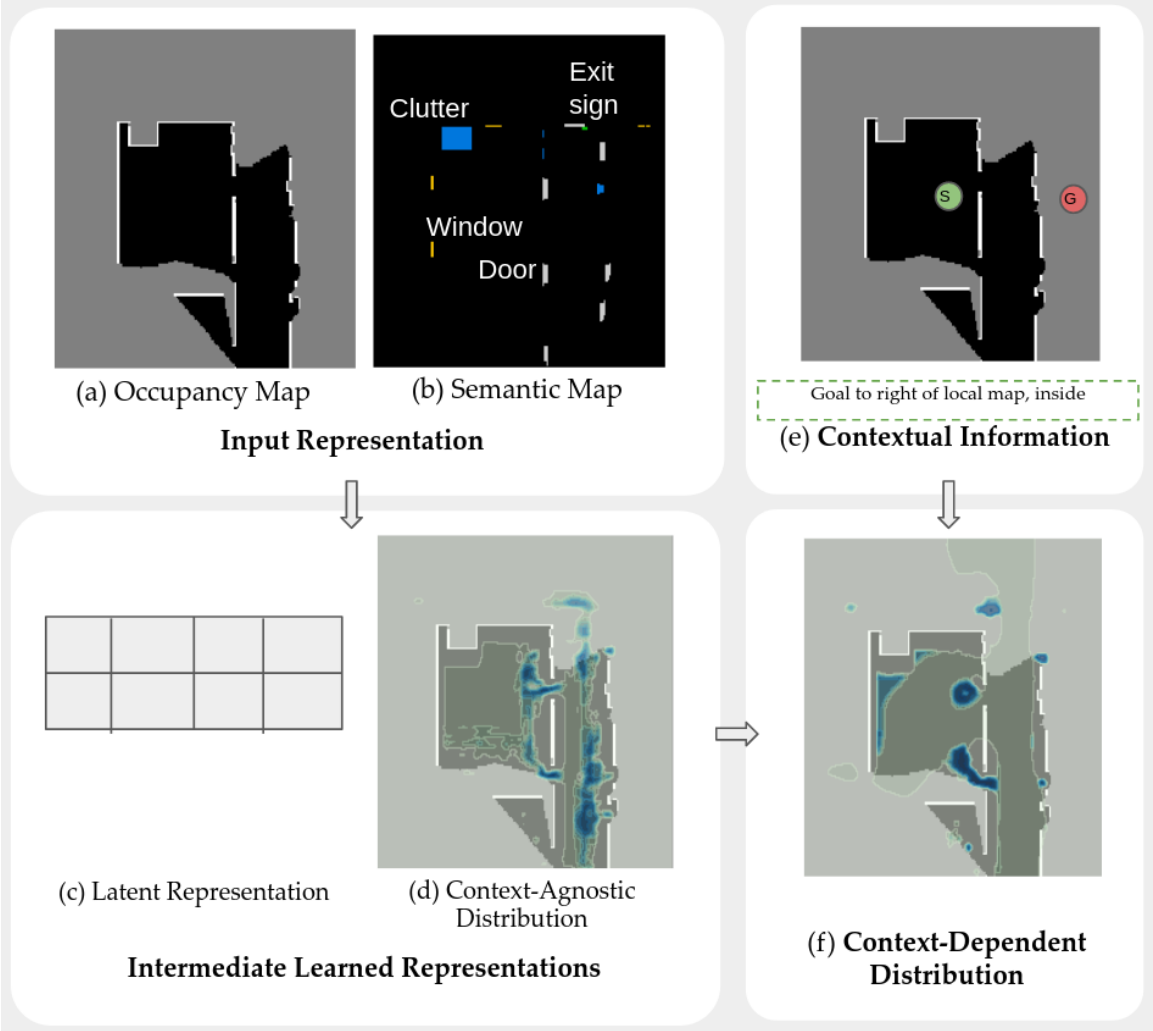


Figure 3-2: **Overview of model used to learn sampling distribution.** Learned distributions are plotted as filled contour plots (blue is high probability, grey-green is low probability) overlaid on occupancy maps. In the first half of the network, local occupancy (a) and semantic (b) maps are passed through a CNN that learns a latent representation (c) and a context-agnostic distribution (d). A second network takes the latent layer (c) and contextual information (e) to learn a task-specific modifier, which is multiplied against the general distribution to obtain a context-dependent distribution (f).

trained model to recover a normalized probability distribution over the current local, partially known map. Grid locations outside of the local sliding window are set to the minimum probability of the predicted window, and the sampling distribution over the entire map is again re-normalized. Finally, we use a Probabilistic Roadmap (PRM) [42] to generate a trajectory from the current robot position to the goal, but modify

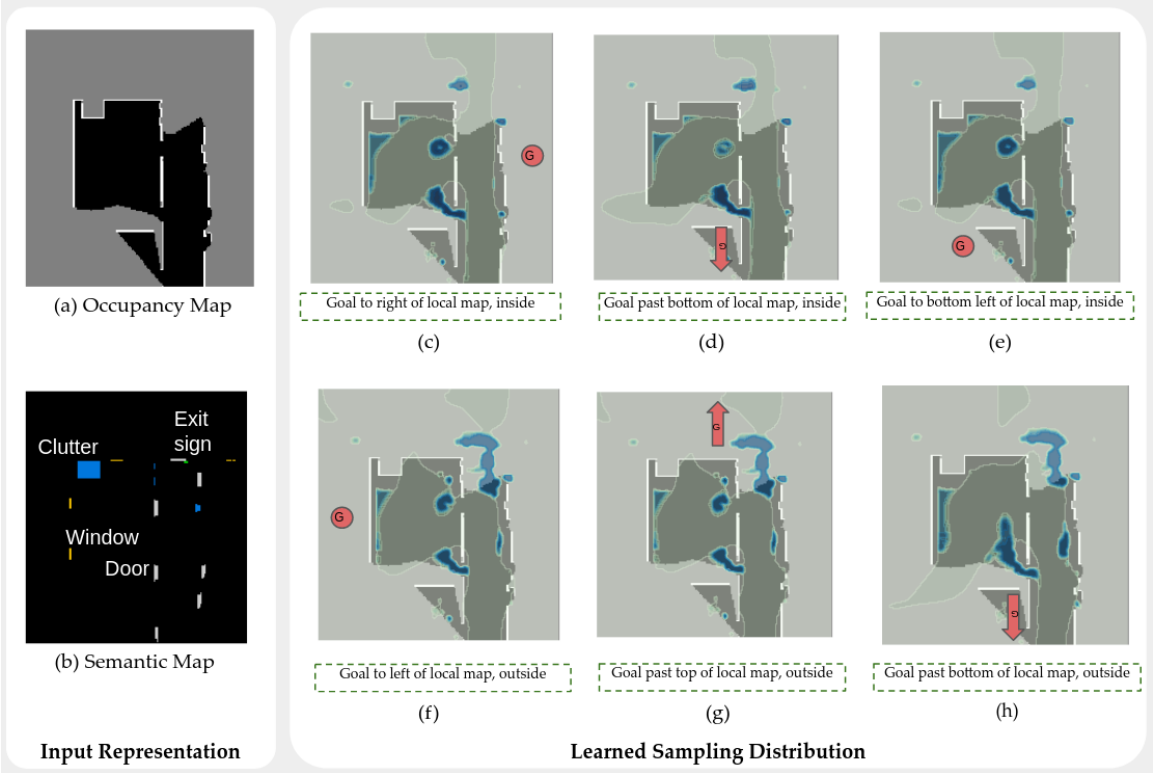


Figure 3-3: **The effects of different contextual inputs on the learned sampling distribution.** Occupancy and semantic maps (a-b) are as in Figure 3-2. Without contextual information, the distribution learns only general navigation heuristics. After adding contextual information about the goal, the network learns different navigation strategies. For example, when the goal is to the right and inside, the exit sign has low probability (c), but when the goal is to the left and outside, the exit sign has high probability, biasing the planner to exit the building (f). The learned distribution is able to place probability near the exit sign, despite not having densely mapped the region.

the planner to sample graph vertices from the learned distribution, as in Equation 3.1, and label graph edges using the probability distribution as an additive cost function, as in Equation 3.2. In practice, we sample nodes from the discretized state $\mathcal{X}_d$ for efficiency, but observe that other sampling methods, such as rejection sampling [54], could be used to sample from the full state space. Similarly, we assume a discretized cost function when determining edge labels, but observe that interpolation methods could be used to generate a cost function over the full state space.

# Chapter 4

# Learned Sampling Distributions Experiments

In this chapter, we demonstrate Learned Sampling Distributions in simulation and on a real-world RC car platform planning in real-time. First, we test the approach in a real-world university floorplan environment in simulation. We demonstrate up to a 2.7x improvement in ability to find feasible plans when limiting the maximum allowable number of planning iterations, and demonstrate up to a 16% reduction in plan cost under some conditions when using our approach as compared to a baseline planner. We conduct an ablation study to determine the relative importance of the learned sampling distribution and the learned cost function. We also demonstrate promising results on a 1/10th scale RC car platform navigating in a building at MIT.

## 4.1 Experimental Setup

We first demonstrated our approach in simulation in a university floorplan environment, which was generated from real-world floorplans of buildings on MIT's campus. Specifically, we used data from 13 floors of 13 unique buildings, split into a train set of 10 buildings and a test set of 3 buildings. For each building, we extracted discrete occupancy maps from floorplan data, and additionally added clutter objects (for example, tables and chairs) to rooms in the floorplans, as in Stein et al. [82]. We also

extracted the locations of doors from floorplan metadata, and generated object-level maps of the environment using the available ground truth door locations, clutter locations, and manually annotated plausible window and exit sign locations. We assumed a holonomic agent equipped with a planar depth sensor with an 85.2 degree field of view and 5 meter range, and an RGB vision system with a 69.4 degree field of view capable of observing objects from 10 meters away[1], with no sensor or pose estimation noise.

## 4.2 Training Dataset Generation

To generate the training dataset $\mathcal{D}$, the collection of trajectories, local body-centered maps, and context, we simulated online planning in the fully known training maps. Specifically, we assumed that the robot had no knowledge of the environment, and simulated greedy planning from random starts to random goals using a modified version of the Open Motion Planning Library's Probabilistic Roadmap Method (PRM) [84]. As the robot planned, it uncovered local occupancy and object-level information from the environment according to its sensor characteristics using raycasting. At each timestep, the robot updated its partial maps of the environment, generated a plan if needed from its current position to the goal using a PRM based on the current partial occupancy map, and took one step along the plan. Replanning was triggered if the newly uncovered geometric information rendered any part of the current plan infeasible, or if 20 steps had been executed without replanning. At approximately every 50 timesteps, we paused online planning and captured the robot's local 160 by 160 pixel occupancy and object-level maps (e.g., $\mathcal{M}$). Then, we randomly generated new goals for the agent, and used an optimal planner[2] and the fully known map of the environment to determine optimal trajectories from the robot's current location

[1] These specifications approximate the performance of the Intel Realsense D345i depth camera [2].

[2] We generate optimal trajectories using the A* algorithm [29] over a discretized grid of the environment, as in practice we find this more efficient than running the sampling-based-motion-planner until asymptotic convergence to an optimal trajectory, but we note that any planner that generates optimal trajectories is sufficient for our purpose. Additionally, we enforce soft costs around obstacles during optimal trajectory generation to encode the desired obstacle clearance behavior.

to the new goals. For each new goal, trajectory elements in the robot's local window, the local hybrid map, the geometric location of the goal, and whether the goal was indoors or outdoors was recorded. The data associated with each new goal constitutes a single training datum, $(\mathcal{T}^*, \mathcal{M}, \mathcal{C})$. The network in Equation 3.14 was optimized using this training data.

## 4.3 Evaluation Pipeline

To evaluate our approach, we simulated over 4000 planning trials with random starts, goals, and initial yaws in the three unseen test maps. We varied the number of planning iterations per query the planner was given to generate a path, denoted as trials with various $N$, and whether or not the robot's goal was indoors or outdoors, denoted as *Inside-Inside* $(II)$ trials and *Inside-Outside* $(IO)$ trials. For each trial, we assumed that the robot had no knowledge of its environment, and simulated planning until the robot reached the goal, failed to find a valid plan at any timestep, or exceeded the maximum number of planning iterations, which was set to 10,000. To simulate planning, a Probabilistic Roadmap (PRM) was generated using a specified sampling distribution and cost function; different test configurations are discussed in Section 4.4. After generating the PRM graph, A* was used to find the lowest cost trajectory from the start to the goal in the PRM graph. The robot then executed the trajectory one step at a time, uncovering geometric and semantic information as it moved; global replanning was triggered if any of the replanning conditions listed in Section 4.2 were met.

## 4.4 Simulation Results

We compared the performance of our approach, which used a learned, context-specific sampling distribution and learned, context-specific cost function $(LSD_s + LSD_s)$, to a baseline approach, which used a uniform sampling distribution and a Euclidean cost function $(Unf + Euc)$. Additionally, we performed an ablation study to quantify the

importance of the learned, context-specific sampling distribution ($LSD_s + Euc$) to the overall performance of the planner.

### 4.4.1 Effects of Learned Sampling Distribution and Learned Cost Function

First, we compared the navigation outcomes of robots navigating using the learned sampling distribution and cost function ($LSD_s + LSD_s$) to the navigation outcomes of robots navigating using a uniform sampling distribution and Euclidean distance cost function ($Unf + Euc$). In general, we observed that the learned sampling distribution placed high probability in hallways and around exit signs, and the learned cost function increased the likelihood of the robot selecting paths that moved through high-probability regions of the environment. A side-by-side comparison of the approaches for a representative $IO$ trial with $N = 500$ is shown in Figure 4-1. In this example, the learned agent placed high probability in the hallway and at the exit sign (a-d). This allowed the learned agent to avoid the overly optimistic navigation behavior of the baseline agent (f-g), which unnecessary detoured into three small rooms prior to reaching the goal. The learned agent reached the goal in 1341 planning iterations, as opposed to the baseline's 1764 planning iterations, and the final learned trajectory was 28.6% shorter than the final baseline trajectory.

Additionally, we compared aggregate navigation results for a variety of test scenarios. We report metrics to address two questions:

1. How likely were the methods to find a plan in a resource-constrained scenario?

2. For trials where both our method and the baseline method find a plan, how costly were the learned plans relative to the baseline plans?

We begun by considering the first question, which addresses the probability of our method finding a valid plan. For each test condition, we calculated the total percentage of trials that succeeded, where success was defined by the agent reaching the goal without failing to find a plan or timing out, which occurred after 10,000 iterations.

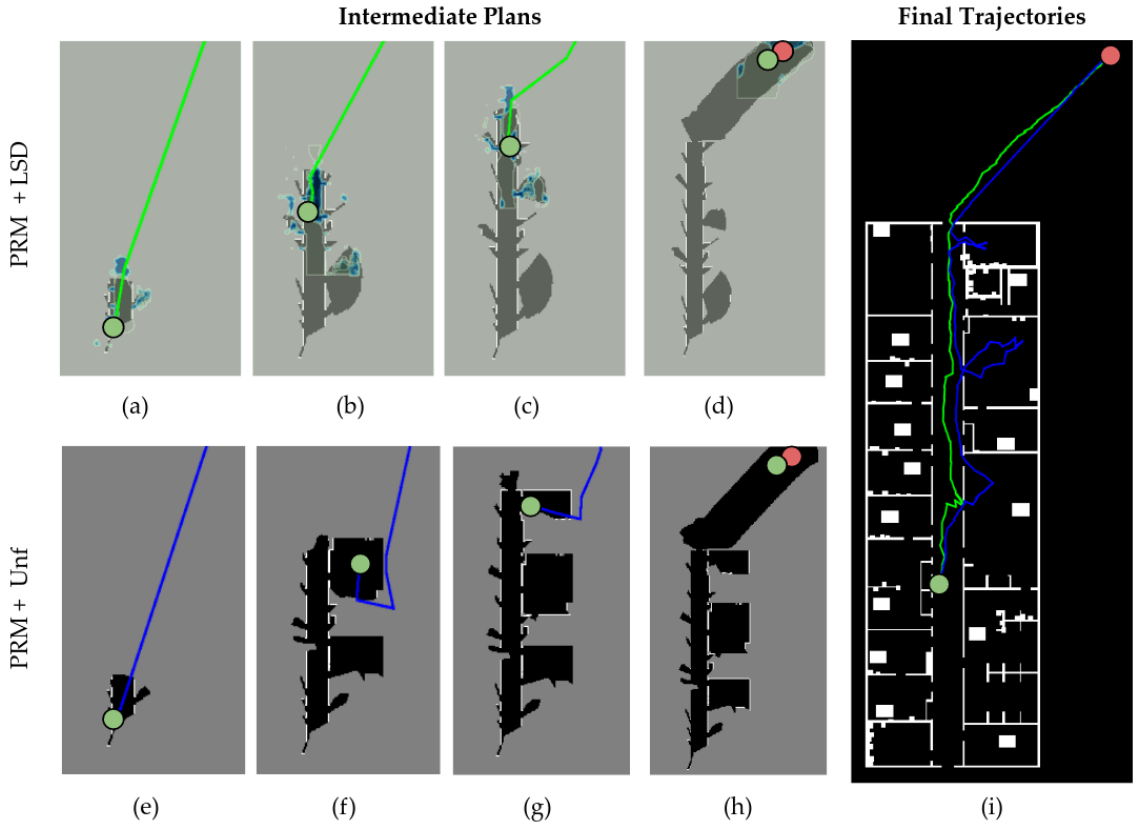


Figure 4-1: **Example intermediate and final trajectories of $LSD_s + LSD_s$ and $Unf + Euc$.** In (a-d), we show the learned sampling distribution and cost function overlaid on the robot's most recent occupancy map (darker blue is higher probability). The goal was set to the top right corner of the map (red circle), and $I_{out}$ indicated that the goal was outdoors. Unlike the baseline (e)-(h), which greedily explored dead-end rooms in the hopes of reaching the goal, the learned distribution largely encouraged the planner to follow the hallway (a)-(d). With the contextual information implicit in the learned sampling distribution, the learned agent traveled 28.6% less distance than the baseline to reach the goal (i).

Results are shown in Figure 4-2. We demonstrated up to a 2.7x increase in ability to find a valid plan over the baseline approach; this demonstrates that our method is more likely to generate random geometric graphs which contain feasible solutions to the planning problem than the baseline approach. Additionally, we note that our method demonstrated the largest gains for low values of $N$, which are analogous to resource-constrained situations. This indicates that the approach has promise for use with resource-constrained vehicles. Finally, we note that our approach demonstrated the largest improvements on the more challenging dataset, *IO*. For example, for the

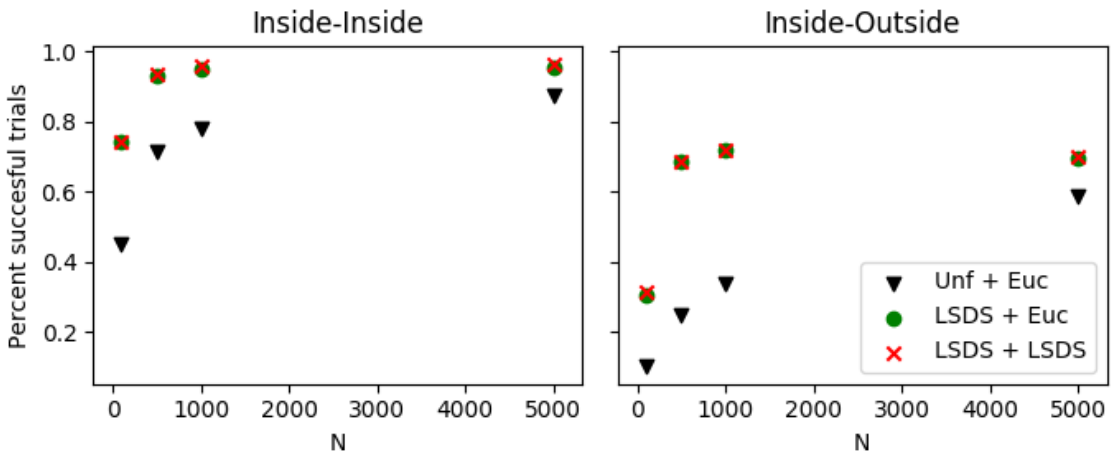


Figure 4-2: **Comparison of plan success rates in MIT Floorplan simulations.** A comparison of plan success rates between $LSD_s + LSD_s$ (red x), $LSD_s + Euc$ (green circle), and $Unf + Euc$ (black triangle) planners, where $N$ is the number of planner iterations per query, which is correlated to the number of samples drawn per step. $LSD_s + LSD_s$ and $LSD_s + Euc$ found plans more frequently than $Unf + Euc$, demonstrating that the learned sampling distribution empowers PRM to find plans more quickly. We observe that the limitations placed on the timesteps per trial may have inhibited the convergence of the harder $IO$ test set. A small percentage ($< 2\%$) of trials were marked as failures due to the vehicle coming into collision with the environment; these trials were removed when calculating success statistics.

$N = 500$ case, $(LSD_s + LSD_s)$ had a 69% success rate, while $(Unf + Euc)$ had a 25% success rate.

Second, we compared the quality of plans generated by the two methods when both methods find plans. We fit a linear regressor with zero intercept to the plan costs of the two methods for mutually successful trials, and demonstrated up to a 16% and 5% improvement in plan cost when using our learned approach as compared to the baseline for the $II$ and $IO$ trials, respectively. We visualize plan costs and linear regressors for all trials in Figure 4-3.

Additionally, we compared the true costs of navigating using our approach and the baseline approach to the cost of optimal navigation in the environment, $\overline{C_l/C^*}$ and $\overline{C_b/C^*}$, respectively. Optimal navigation costs were determined by planning using an optimal planner with soft costs in the fully known map. We report the mean and standard error of the mean of these quantities, along with additional metrics, in Table 4.1. We note that our method provided modest improvements in average plan cost in

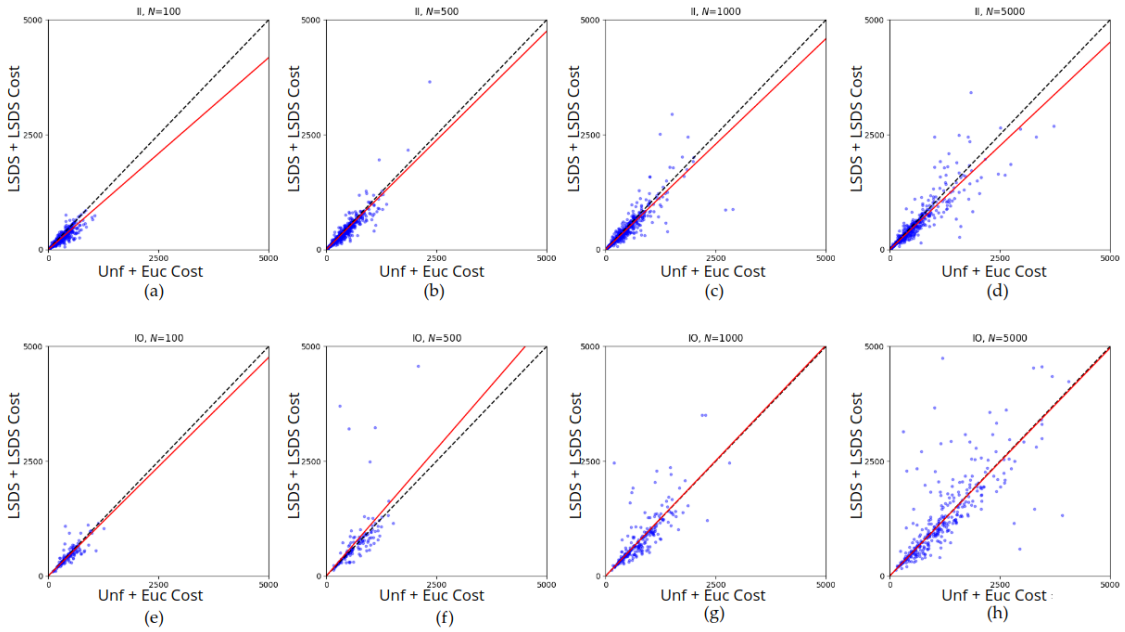


Figure 4-3: **Comparison of distance traveled in MIT Floorplan simulations.** Scatter plots of distance travelled in simulation units for $LSD_s + LSD_s$ vs. $Unf + Euc$ (blue circles) and the slope calculated by linear regression (red line) for various test conditions, where $N$ is the number of planner iterations per query, as in Figure 4-2. The black dashed line is plotted as a reference for equal cost.

some cases, and did not decrease performance in a statistically significant way for any of the cases. This indicates that our approach does not negatively effect the cost of planning for any trials, while significantly increasing the probability that the planner successfully finds a plan.

Finally, we considered additional representative comparisons of navigation results using our learned approach as compared to the baseline approach in Figure 4-4. In many cases (a, b, c, i, j, l), the baseline algorithm optimistically entered rooms and failed to find plans, while the learned method largely avoided unnecessary rooms and successfully completed trials. However, as unknown space is inherently uncertain, the learned agent sometimes entered unnecessary rooms; in many of these cases (b, c, i, j), the agent was able to quickly exit the room by placing high probability at the doorway of the room, and successfully completed the trial. In other cases (f, g, h), while the baseline was able to find a plan, the learned planner avoided unnecessary exploration and found a lower cost plan than the baseline. Occasionally (d, e), the

| **Dataset** | ***II*** | ***II*** | ***II*** | ***II*** | ***IO*** | ***IO*** | ***IO*** | ***IO*** |
|---|---|---|---|---|---|---|---|---|
| **N** | 100 | 500 | 1000 | 5000 | 100 | 500 | 1000 | 5000 |
| $\bar{\boldsymbol{C}}$ | 0.84 | 0.95 | 0.92 | 0.90 | 0.95 | 1.11 | 1.01 | 0.99 |
| $\|\|\boldsymbol{D}\|\|$ | 1500 | 1162 | 1016 | 783 | 1500 | 550 | 705 | 595 |
| $\|\|\boldsymbol{D_m}\|\|$ | 642 | 819 | 787 | 669 | 140 | 132 | 229 | 315 |
| $\boldsymbol{R^2}$ Score | 0.81 | 0.87 | 0.77 | 0.82 | 0.67 | 0.34 | 0.63 | 0.59 |
| $\overline{\boldsymbol{C_l/C^*}}$ | 1.21 $\pm$0.01 | 1.24 $\pm$0.01 | 1.31 $\pm$0.02 | 1.42 $\pm$0.03 | 1.22 $\pm$0.03 | 1.73 $\pm$0.16 | 1.75 $\pm$0.09 | 2.41 $\pm$0.09 |
| $\overline{\boldsymbol{C_b/C^*}}$ | 1.40 $\pm$0.02 | 1.32 $\pm$0.01 | 1.38 $\pm$0.02 | 1.51 $\pm$0.03 | 1.24 $\pm$0.02 | 1.51 $\pm$0.06 | 1.69 $\pm$0.07 | 2.35 $\pm$0.08 |

Table 4.1: **A comparison of plan costs between $LSD_s$+$LSD_s$ and $Unf$+$Euc$.** A comparison of plan costs between $LSD_s + LSD_s$ and $Unf + Euc$ when both planners succeeded, broken out by dataset and planner iterations per query ($N$), which is correlated to the number of samples drawn per step. $||D||$ and $||D_m||$ are the total number of trials run and the number of trials where both planners succeeded. $||D||$ varies over $N$ due to the time involved in running longer trials. $\bar{C}$ and $R^2$ score are the slope and fit score of a linear regression with zero intercept. $\overline{C_l/C^*}$ and $\overline{C_b/C^*}$ are the mean and standard error of the mean of the trajectory cost divided by the resolution-optimal trajectory cost for learned and baseline respectively. In the *II* dataset, $LSD_s + LSD_s$ found lower cost plans than $Unf + Euc$, indicating that using the learned sampling distribution and cost function with our chosen SBMP resulted in better navigation outcomes.

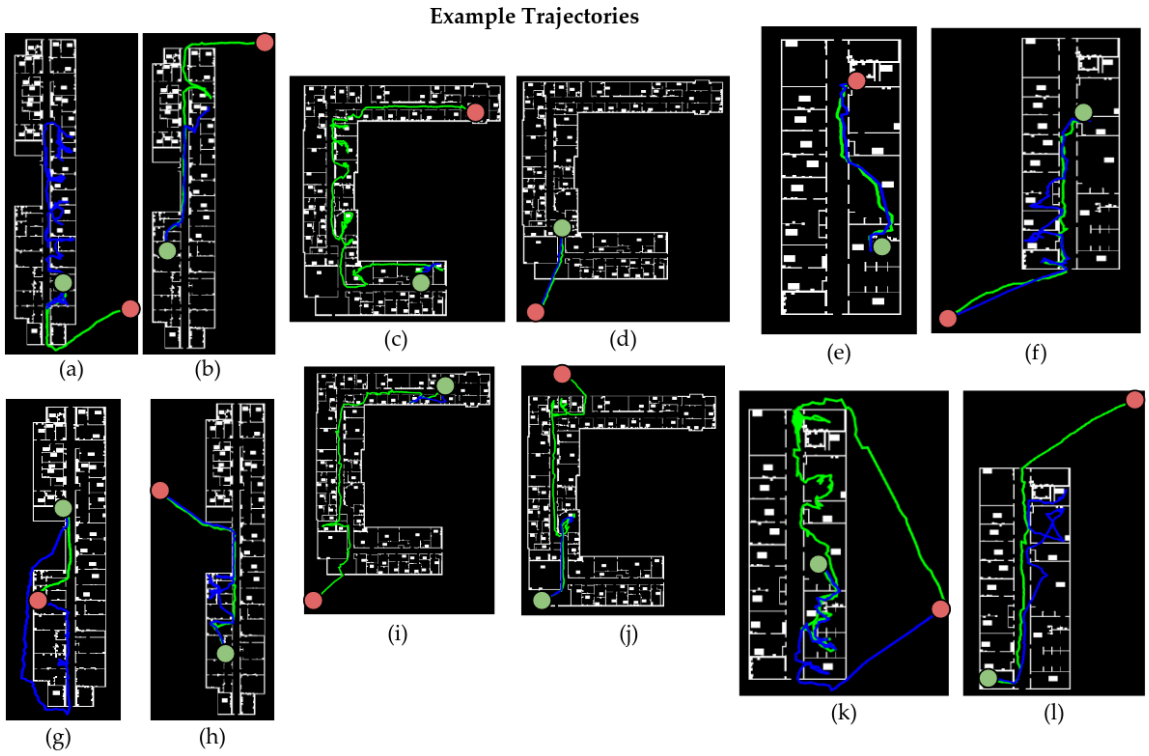


Figure 4-4: **Additional comparisons of $LSD_s + LSD_s$ and $Unf + Euc$.** In (a)-(l), we show example final trajectories of $LSD_s + LSD_s$ (green) and $Unf + Euc$ (blue) traversing from start (green circle) to goal (red circle). For example, in (f), both the learned and baseline approaches mistakenly entered rooms prior to reaching the goal, but the learned approach avoided more rooms than the baseline approach, resulting in a lower cost plan. In (i), the baseline approach failed while trying to exit a room, but the learned approach largely remained in hallways and reached the goal.

learned and baseline planners found similar cost plans; this occurred most commonly for short plans. Finally, in some uncommon cases (k), the baseline planner found a shorter plan than the learned planner.

### 4.4.2 Ablation Study: Learned Sampling Distribution and Euclidean Distance Cost Function

In Section 4.4.1, we demonstrated the benefits of using sampling distributions and cost functions informed by geometric and object-level information for robot navigation in unknown environments. By enabling the agent to reason about both where to place samples and how to weigh edges in unknown environments, we enabled the

| **Dataset** | ***II*** | ***II*** | ***II*** | ***II*** | ***IO*** | ***IO*** | ***IO*** | ***IO*** |
|---|---|---|---|---|---|---|---|---|
| **N** | 100 | 500 | 1000 | 5000 | 100 | 500 | 1000 | 5000 |
| $\bar{\boldsymbol{C}}$ | 0.82 | 0.94 | 0.94 | 0.94 | 0.96 | 1.01 | 0.97 | 0.98 |
| $\|\|\boldsymbol{D}\|\|$ | 1500 | 1500 | 1500 | 1500 | 1500 | 1313 | 982 | 634 |
| $\|\|\boldsymbol{D_m}\|\|$ | 627 | 1078 | 1182 | 1297 | 128 | 328 | 295 | 330 |
| $\boldsymbol{R^2}$ Score | 0.73 | 0.85 | 0.84 | 0.89 | 0.83 | 0.56 | 0.54 | 0.75 |
| $\overline{\boldsymbol{C_l/C^*}}$ | 1.20 $\pm$0.01 | 1.26 $\pm$0.01 | 1.33 $\pm$0.03 | 1.44 $\pm$0.03 | 1.18 $\pm$0.02 | 1.46 $\pm$0.03 | 1.67 $\pm$0.06 | 2.30 $\pm$0.09 |
| $\overline{\boldsymbol{C_b/C^*}}$ | 1.39 $\pm$0.02 | 1.33 $\pm$0.01 | 1.40 $\pm$0.03 | 1.51 $\pm$0.03 | 1.22 $\pm$0.02 | 1.44 $\pm$0.03 | 1.68 $\pm$0.05 | 2.34 $\pm$0.09 |

Table 4.2: **A comparison of plan costs between $LSD_s + Euc$ and $Unf + Euc$.** A comparison of plan costs between $LSD_s + Euc$ and $Unf + Euc$ when both planners succeed, broken out by dataset and planner iterations per query ($N$). Planner metrics are as in Table 4.1. In the *II* dataset, $LSD_s + Euc$ found lower cost plans than $Unf + Euc$, indicating that using the learned sampling distribution with our chosen SBMP results in better navigation outcomes, even when a Euclidean distance cost function is used.

agent to generate informed plans in unknown spaces. However, for the purpose of informing future research, we also conducted an ablation study, in which we considered the effects of using the learned sampling distribution with a Euclidean distance cost function, $LSD_s + Euc$. Complete results for this trial are shown in Table 5.1.

First, we compared the success rate of $LSD_s + Euc$ to the success rates of $Unf + Euc$ and $LSD_s + LSD_s$. Plan success rates are shown in Figure 4-2; $LSD_s + Euc$ was more likely to find a plan than $Unf + Euc$, and performed comparably to $LSD_s + LSD_s$. This was the expected behavior, as the sampling distribution, not the cost function, modifies the connectivity of the graph over which we consider solutions to the optimization in Equation 2.5. Because both planners using the $LSD_s$ sampling distribution used geometric and object-level information to inform the structure of the graphs over which planning occurs, they were more likely to find solutions to the optimization than the baseline planner.

Next, we compared the relative costs of plans generated using $LSD_s + LSD_s$ and $LSD_s + Euc$; results are reported in Tables 4.1 and 5.1. When comparing the costs of plans generated by the two planners to costs of optimal plans in the environment,

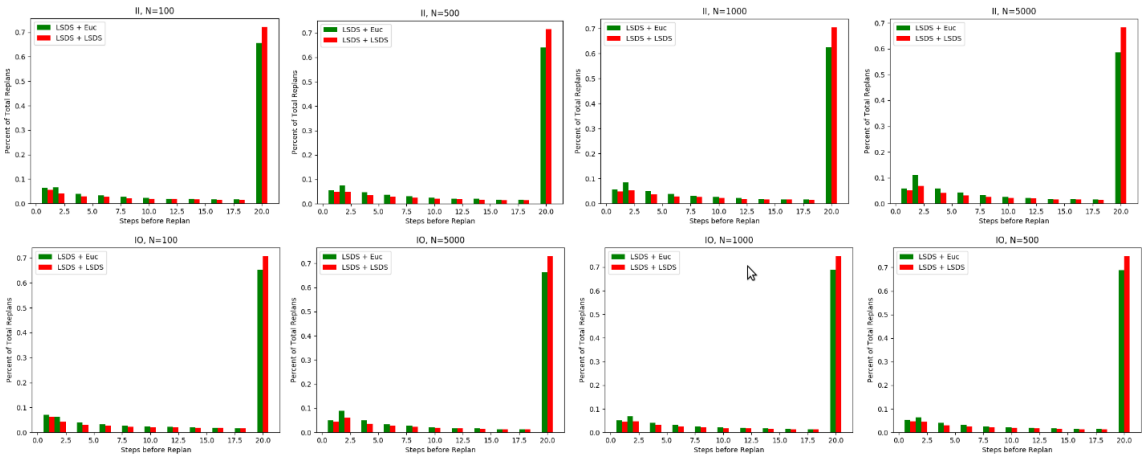


Figure 4-5: **Comparison of replanning rates.** Comparison of replanning rates of $LSD_s$+$LSD_s$ and $LSD_s$+$Euc$ for different trials. $LSD_s$+$LSD_s$ (red) was more likely to generate plans that remained feasible over time, indicated by the larger number of plans replanned after 20 timesteps (far right bar of each plot), which occurred due to a user-enforced timeout. $LSD_s + Euc$ was more likely to generate plans that became infeasible shortly after planning, indicated by a larger percentage of replans occurring after less than 20 timesteps. Specifically, $LSD_s + Euc$ generated many plans that were determined invalid after 1-10 timesteps.

assuming soft costs, we demonstrated no significant difference in plan cost between the two approaches. However, upon further inspection, the two methods exhibited different planning behaviors; specifically, $LSD_s + Euc$ was more likely to generate plans that were later observed to be in collision than $LSD_s + LSD_s$. To quantify this effect, we analyzed the number of times that replanning was triggered in the planning loop for the two approaches. Specifically, we compared the replan rates of the two methods; this metric indicates the number of iterations it took for a plan to become invalid, once found. We found that $LSD_s + Euc$ was more likely than $LSD_s + LSD_s$ to generate infeasible paths and require replanning prior to the timeout, as shown in Figure 4-5. This indicates that the learned cost function enables the sampling-based motion planner to select promising trajectories in unknown environments.

## 4.5 Real-World Navigation Results

We also demonstrated our approach on a 1/10th scale RC car platform completing a real-world navigation task in a building at MIT. The car was equipped with an Intel D435i Depth Camera [2], an Intel T265 Tracking Module [3], and an Intel Nuc i7 [1]. Robot poses were generated using the tracking module. Occupancy maps were generated online from depth images using OctoMap; object-level maps were generated online from RGB images with object detections using an object-level mapping system, Robust Object-based SLAM for High-speed Autonomous Navigation (ROSHAN) [65]. Object detections for ROSHAN were generated online using an object detector based on SSD-Mobilenet [33, 52, 4] and fine-tuned to detect windows and doors using the OpenImages dataset [47]. Exit signs were detected using an HSV filter. Once 3D object ellipsoids were populated in the 3D object-level map, object ellipsoids were approximately projected onto the 2D ground plane to generate 2D object-level maps. We ran our approach on real-world data using the neural network trained in simulation; no additional fine-tuning was required.

We performed two comparisons of our approach on real-world data. First, we generated an offline dataset of the robot navigating in a hallway at MIT, then qualitatively compared the instantaneous plans generated by $LSD_s{+}LSD_s$ and $Unf{+}Euc$ on the dataset. Second, we demonstrated our approach in-the-loop for a challenging navigation task in a building at MIT, and demonstrated improved navigation performance as compared to the baseline approach.

### 4.5.1 Offline Comparison of Plans

We qualitatively compared instantaneous plans generated using the $LSD_s{+}LSD_s$ and $Unf{+}Euc$ planners on an offline dataset of the robot navigating in a hallway at MIT. The learned distribution placed high probability in the hallway and door regions of the environment, and the learned planner generated plans that exited through the door at the end of the hallway, while the baseline approach generated plans that attempted to exit the hallway by turning around and exiting through an unmapped portion of

the wall. This qualitative behavior indicated that the learned sampling distribution and cost function could be used to inform intelligent navigation behaviors. Example offline trajectories are shown in Figure 4.5.1.

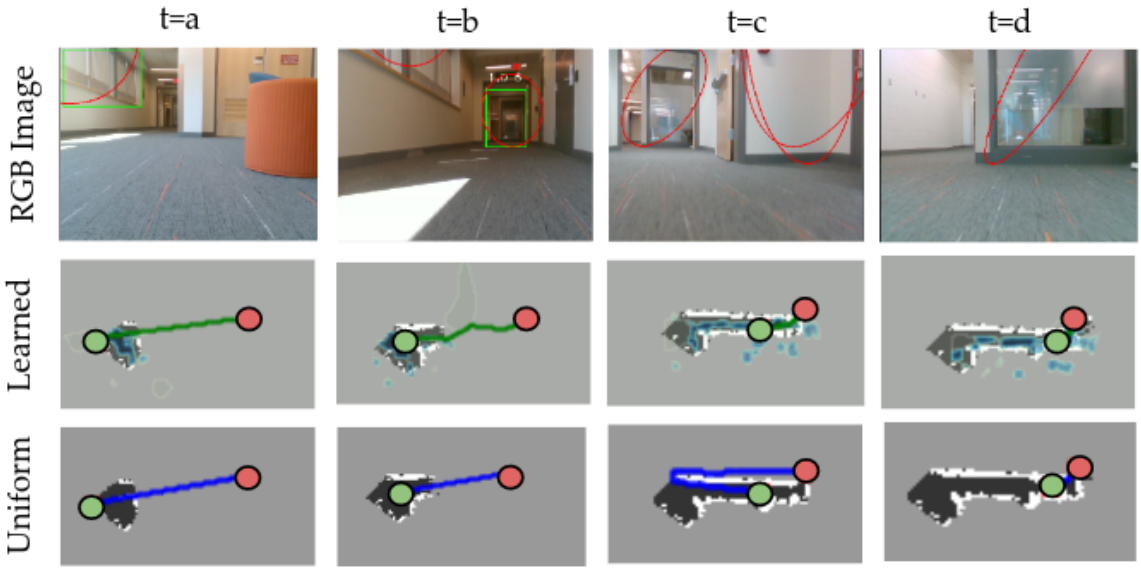


Figure 4-6: **Qualitative comparison of plans on real-world data.** Qualitative comparison of plans generated by $LSD_s + LSD_s$ and $Unf + Euc$ during an offline planning trial. $LSD_s + LSD_s$ placed high probability in the hallway and doorway regions, and generated plans that navigated directly down the hallway towards the goal, while $Unf + Euc$ attempted to plan through an unmapped portion of the wall (t=c). All images are approximately aligned.

### 4.5.2 Online Comparison of Plans

We tested our approach on a navigation task in a building at MIT, with $N = 500$ and trajectory following using pure pursuit. Despite a slower planning loop (approximately 2 Hz), $LSD_s + LSD_s$ quickly identified doors and an exit sign and planned down the hallway towards the goal. $Unf + Euc$, which planned at approximately 6 Hz, exhibited severe thrashing behavior and attempted to greedily plan through a wall towards the goal. After being manually backed up by an operator to allow for progress, the baseline planner continued down the hallway in the wrong direction in an attempt to greedily minimize its distance to the goal, and ultimately failed to find a valid plan. Example trajectories from the two trials are shown in Figure 4.5.2.

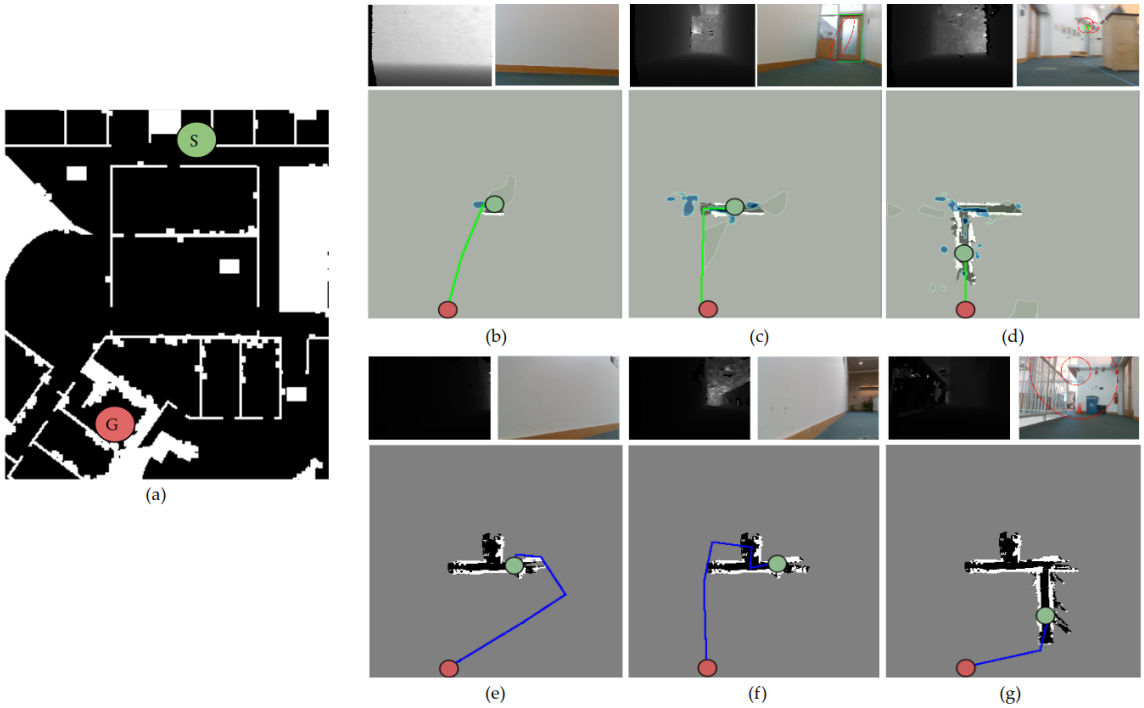


Figure 4-7: **Qualitative comparison of plans generated during online planning.** Qualitative comparison of plans generated by $LSD_s + LSD_s$ and $Unf + Euc$ during an online planning trial. An occupancy map of the environment and approximate start and goal locations are shown in (a). In (b-g), we show onboard depth images, RGB images with object detections, and current robot plans overlaid on the robot's current occupancy map for six timesteps during planning. For the $LSD_s + LSD_s$ cases, we also plot the current learned sampling distribution over the occupancy map. $LSD_s + LSD_s$ (b-d) quickly identified hallways using geometric structure and semantic information about the locations of doors and exit signs, and successfully navigated to the goal. $Unf + Euc$ (e-g) exhibited severe thrashing behavior, attempted to navigate through noisy walls, and ultimately navigated into an adjacent hallway that did not lead to the goal and failed to find a plan.

## 4.6 Conclusion

In this chapter, we evaluated Learned Sampling Distributions, a novel sampling-based motion planning algorithm that enables efficient navigation in unknown environments using hybrid geometric and semantic information. We evaluated the approach in simulation in a real-world university floorplan environment, and demonstrated that our approach was up to 2.7 times more likely to find a valid plan than a baseline planner, without sacrificing plan quality. We also demonstrated promising real-world performance on a 1/10th scale RC car planning online in a building at MIT. These results

indicate that hybrid geometric and semantic information can be used to improve navigation results in unknown environments.

However, one of the drawbacks of Learned Sampling Distributions is that the learned sampler is only able to inform intelligent navigation strategies in a local window. Beyond the local window, the sampler returns to greedy behavior. In practice, the size of the local window is limited by the computational cost of calculating the probability distribution, based on high-resolution geometric and semantic information, over the high-resolution hybrid maps. In Chapter 5, we discuss a preliminary approach that uses coarse-resolution overhead images to approximate cost and traversability functions for hierarchical, perceptually informed navigation in long length scale environments.

# Chapter 5

# Perceptually Informed Abstractions for Efficient Robot Navigation

In this chapter, we present Perceptually Informed Abstractions, a preliminary hierarchical planning approach that enables efficient robot navigation in long length scale environments. We review the optimal planning problem and motivate the use of a multi-resolution hierarchical decomposition of the state space, informed by non-geometric information, to inform efficient hierarchical planning. We model cost and traversability functions of low resolution plans as distributions conditioned on low resolution overhead images of the environment, and discuss a method for learning the parameters of the distributions using a convolutional neural network (CNN).

## 5.1 Problem Formulation

In section 2.5, we motivated the use of hierarchical models to increase planning tractability in long length scale environments. Specifically, we noted that *abstract* plans $\mathbf{p}$, composed of abstract actions $\mathbf{a}$ defined in a reduced model of the planning problem, and informed by abstract cost and validity functions $\mathbf{c}(\mathbf{a})$ and $\mathbf{f}(\mathbf{a})$, can be used to inform the search for a *primitive* plan in long-length scale environments. The process of generating a primitive plan informed by an abstract plan is called plan *refinement*.

While abstractions are a convenient way to reduce the size of the search space for planning, abstractions are only useful if they capture the properties of the configuration space $\mathcal{X}$ necessary to inform the search for primitive solutions $p$. Intuitively, good choices of abstract actions, abstract cost functions, and abstract validity functions are low-dimensional and simple to compute, but also capture the properties of the original planning problem that inform low cost planning behaviors. One common approach to abstract planning is using the underlying geometry of the state space to define compact regions of the environment (e.g., the constrained Delaunay triangulation over an occupancy map [90], signal compression using information bottleneck over an occupancy quadtree [48], convolution and max pooling over a height map [44]) that can be processed to generate low-dimensional abstract action spaces, abstract cost functions, and abstract validity functions. However, in long length scale environments, it can be expensive to directly compute an abstraction based on dense geometry.

## 5.2 Perceptual Information for Navigation

Defining abstract actions and calculating their properties from dense geometry can be expensive, but geometry is not the only type of information that can be used to inform intelligent planning decisions. In this work, we propose using non-geometric information to inform abstract planning in a hierarchical planner. Specifically, we would like to take advantage of low resolution overhead imagery, such as satellite imagery, which has been demonstrated to admit useful navigation cues [79, 61, 62, 20], to inform abstract planning. For example, consider a ground vehicle navigating outdoors in an environment with a forest and a field (Figure 5-1a). We would like for the robot to use low resolution satellite imagery to infer that paths through the forest are likely to be higher cost than paths through the field. Similarly, consider a ground vehicle navigating in an environment with a lake surrounded by a forest (Figure 5-1b). We would like for the robot to quickly identify the lake as an untraversable area, and focus its computation on finding a valid plan through the challenging forest

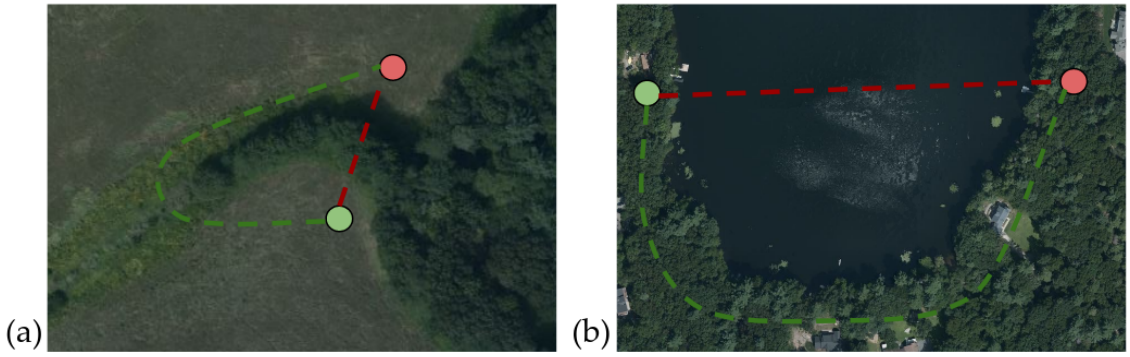


Figure 5-1: **Two examples where overhead images can inform intelligent navigation behaviors.** The robot is assumed to be a ground vehicle, and the robot start and goal locations are shown as green and red circles, respectively. (a) The robot navigates in an environment with a forest and a field. We would like for the hierarchical planner to infer that paths through the forest (red dashed line) are likely to have high cost primitive refinements and select longer abstract plans that remain in the low cost field (green dashed line). (b) The robot navigates around a lake surrounded by a forest. We would like for the hierarchical planner to quickly prune infeasible paths through the water (red dashed line) and focus computation on finding a valid plan through the challenging forest terrain (green dashed line). Images collected from Bing Maps [58].

terrain. In this work, we refer to this form of non-geometric reasoning as *perceptual* reasoning. We hypothesize that planning using perceptually informed abstractions can be more efficient than planning using geometrically informed abstractions in some environments.

To enable perceptual reasoning in long length scale environments, we define an abstract action space using a simple region-based decomposition technique that does not rely on the dense geometry of the environment. Then, we learn abstract cost and validity functions over the abstract action space, conditioned on low-resolution satellite imagery, to encode intelligent navigation behaviors. We use the abstract action space, cost function, and validity function to generate solutions to the optimization in Equation 2.6. Finally, we discuss a method for incorporating perceptually informed abstractions into a hierarchical planning pipeline.

## 5.3 Perceptually Informed Abstractions for Navigation

Formally, we assume that the configuration space $\mathcal{X}$ can be decomposed into a finite set of regions $\mathcal{R}$[1]. We define an abstract action $\mathbf{a}_i$ as the set of primitive actions that begin in and remain in region $\mathcal{R}_j$ until terminating in region $\mathcal{R}_k$, similar to Vega-Brown and Roy [90],

$$\mathbf{a}_i = \{p | p \in \mathcal{X};\ p_t \in \mathcal{R}_j\ \forall t \in \{0, 1, ..., n-2\}; p_{n-1} \in \mathcal{R}_k\}. \tag{5.1}$$

To use this definition of abstract actions to generate solutions to the optimization in Equation 2.6, two properties of the abstract actions must be computed. First, each action must be classified as valid or invalid. The goal of the abstract validity function $\mathbf{f}(\mathbf{a})$ is to determine if a valid primitive refinement of an abstract action exists for a particular planning instance. This function allows the planner to quickly prune regions of the action space that cannot be refined into a valid primitive plan. In practice, directly determining the validity of an abstract action can require searching over all possible primitive refinements of the abstract action, which is computationally expensive. However, we recognize that many environments admit noisy perceptual cues that can be used to approximate the validity of abstract actions, as in the lake example in Figure 5-1b. To model these cues, we define the validity of an abstract action $\mathbf{a}_i$ as a draw from a distribution which encodes the probability that an abstract action is traversable, conditioned on an overhead image,

$$\mathbf{f}(\mathbf{a}_i; M_p, \beta_i) \sim \delta_{trav}(\beta_i; M_p), \tag{5.2}$$

where $M_p$ is a low resolution overhead image of the environment that has been metrically aligned with the region-based decomposition, and $\delta_{trav}$ is a probability distri-

[1]For simplicity of implementation, we assume that regions are defined by decomposing the environment into low-resolution grid cells, but we note that other decomposition functions could be used.

bution with abstract action-specific parameters $\beta_i$. For the overhead image problem, we will assume $\mathbf{f}(\mathbf{a})$ is a traversability function, but note that other forms of action validity can be encoded.

Second, we define the abstract cost function $\mathbf{c}(\mathbf{a})$. The goal of the abstract cost function is to approximate the costs of primitive refinements of abstract actions. By modeling the costs of abstract actions, we enable the hierarchical planner to focus planning on abstract actions that are likely to lead to low-cost primitive plans. Because the true cost of a primitive action guided by an abstract action is dependent on the underlying geometry of $\mathcal{X}$ and the primitive locations where the robot begins and ends the abstract action, it is not obvious how to calculate the ground truth cost of an abstract action without conditioning on other trajectory elements and the ground truth geometry. However, we again recognize that many environments admit noisy, imperfect perceptual cues that can be used to approximate costs of abstract actions, as in the forest . To capture the uncertainty in these cues, we model the cost of each abstract action as a draw from a probability distribution conditioned on an overhead image,

$$\mathbf{c}(\mathbf{a}_i; M_p, \alpha_i) \sim \delta_{cost}(\alpha_i; M_p), \tag{5.3}$$

where $\delta_{cost}$ is a probability distribution with abstract action-specific parameters $\alpha_i$. Under this model, abstract actions with higher costs are more likely to have high cost primitive refinements, while abstract actions with lower costs are more likely to have low cost primitive refinements.

While the definitions of $\mathbf{f}(\mathbf{a})$ and $\mathbf{c}(\mathbf{a})$ can accept any distribution $\delta_{trav}$ and $\delta_{cost}$ parameterized by a finite number of parameters $\beta$ and $\alpha$, in practice, it is expensive to reason over the space of all possible probability distributions. In the sections that follow, we choose to model the traversability distribution $\delta_{trav}$ as a Bernoulli distribution with parameter $\beta = \{\pi\}$,

$$\mathbf{f}(\mathbf{a}_i; M_p, \beta_i) \sim \text{Bern}(\pi_i; M_p) \tag{5.4}$$

Additionally, we choose to model the cost distribution $\delta_{cost}$ as a Gaussian distribution

with parameter $\alpha = \{\mu\}$ and constant, hand-tuned standard deviation $\sigma^2$,

$$\mathbf{c}(\mathbf{a}_i; M_p, \alpha_i) \sim \mathcal{N}(\mu_i, \sigma; M_p) \tag{5.5}$$

## 5.4 Learned Abstract Functions

While we intuitively observe that $M_p$ contains navigation cues that can inform the abstract cost and traversability functions in Equations 5.2 and 5.3, it is not obvious how to define a mapping from a noisy, low-resolution pixel space to the space of abstract function parameters. However, determining example image-cost function and image-traversability function value pairs is a more tractable endeavor. We propose learning the parameters of the functions in Equations 5.2 and 5.3 by maximizing the probability of recovering image-function value pairs over a training dataset.

Formally, we assume a dataset $\mathcal{D}$ that includes $z$ tuples of overhead images, geometric maps, and optimal primitive trajectories,

$$\mathcal{D} = \{(M_p^0, M_g^0, \mathcal{T}^0), (M_p^1, M_g^1, \mathcal{T}^1), ..., (M_p^z, M_g^z, \mathcal{T}^z)\}. \tag{5.6}$$

Additionally, we assume that primitive feasibility and cost functions, $f(a)$ and $c(a)$, are known in the training environments.

### 5.4.1 Learned Traversability

To learn the abstract traversability function, we note that we can label abstract action $\mathbf{a}_i$ as feasible if there exists a primitive plan that traverses through region $\mathcal{R}_j$ into region $\mathcal{R}_k$. For each training datum in $\mathcal{D}$, we generate a ground truth traversability label $M^s$ by enumerating all possible refinements of all possible abstract actions in $M_g$ and recording the binary traversability label $\mathbf{l}_i$, which indicates whether or not a feasible primitive refinement of each abstract action exists, that is, the label indicates whether any primitive plan exists that satisfies the constraints of the abstract action

[2]In future work, we would also like to learn $\sigma$.

definition in Equation 5.1. We propose learning the abstract traversability function by maximizing the likelihood of recovering the ground truth traversability of each abstract action $\mathbf{l}_i$ in the training dataset,

$$\arg\max_{\beta} \sum_{\zeta=0}^{|\mathcal{D}|-1} \sum_{i=0}^{|M_\zeta^s|} p(\mathbf{f}(\mathbf{a}_i; M_p, \beta_i) = \mathbf{l}_i). \tag{5.7}$$

For numerical stability, we minimize the negative log likelihood of Equation 5.7,

$$\arg\min_{\beta} \sum_{\zeta=0}^{|\mathcal{D}|-1} \sum_{i=0}^{|M_\zeta^s|} -\log\ p(\mathbf{f}(\mathbf{a}_i; M_p, \beta_i) = \mathbf{l}_i). \tag{5.8}$$

Plugging in the Bernoulli distribution as the form of $p(\mathbf{f}(\mathbf{a}_i; M_p, \beta_i))$, we recover the binary cross entropy loss function as the objective of the optimization,

$$\arg\min_{\pi} \sum_{\zeta=0}^{|\mathcal{D}|-1} \sum_{i=0}^{|M_\zeta^s|} -\mathbf{l}_i \log\ \pi_i - (1-\mathbf{l}_i)\log\ (1-\pi_i)). \tag{5.9}$$

### 5.4.2 Learned Cost Function

To learn the abstract cost function, we note that we can treat the cost of an example traversal $\mathcal{T}^s$ from region $\mathcal{R}_j$ to $\mathcal{R}_k$ as an example draw of the abstract cost function $\mathbf{c}(\mathbf{a}_i; M_p, \alpha_i)$. Formally, we calculate the ground truth cost of a sample abstract action $\mathbf{a}_i^s$ as the sum of consecutive primitive actions that occur in $\mathcal{R}_j$ before transitioning to $\mathcal{R}_k$ in example trajectory $\mathcal{T}^s$,

$$\boldsymbol{c}(\mathbf{a}_i^s) = \sum_{a \in \mathcal{F}} c(a), \tag{5.10}$$

where $\mathcal{F}$ is the set of primitive actions $a$ in trajectory $\mathcal{T}^s$ that occur in region $\mathcal{R}_j$ before transitioning to $\mathcal{R}_k$, i.e., $\mathcal{F} = \{a | a \in \mathcal{T}^s;\ a_t \in \mathcal{R}_j\ \forall t \in \{l, l+1, ..., m-1\}; a_m \in \mathcal{R}_k\}$[3].

We propose learning the abstract cost function by maximizing the likelihood of recovering the costs of sample abstract actions $\boldsymbol{c}(\mathbf{a}_i^s)$ in the training dataset,

[3]For non-convex environments, there may be multiple samples of a particular abstract action in a given trajectory. In this case, we record both sample actions separately.

$$\arg\max_{\alpha} \sum_{\zeta=0}^{|\mathcal{D}|-1} \sum_{i=0}^{n^s-1} p(\mathbf{c}(\mathbf{a}_i; M_p, \alpha_i) = \mathbf{c}(\mathbf{a}_i^s)). \tag{5.11}$$

where $n^s$ is the number of sample abstract actions in sample trajectory $\mathcal{T}_\zeta^s$.For numerical stability, we minimize the negative log likelihood of Equation 5.11,

$$\arg\min_{\alpha} \sum_{\zeta=0}^{|\mathcal{D}|-1} \sum_{i=0}^{n^s-1} -\log\ p(\mathbf{c}(\mathbf{a}_i; M_p, \alpha_i) = \mathbf{c}(\mathbf{a}_i^s)). \tag{5.12}$$

Plugging in the Gaussian distribution as the form of $\mathbf{c}(\mathbf{a})$, our optimization becomes:

$$\arg\min_{\mu} \sum_{\zeta=0}^{|\mathcal{D}|-1} \sum_{i=0}^{n^s-1} -\log\ (\mathcal{N}(\mathbf{c}(\mathbf{a}_i^s); \mu_i, \sigma)). \tag{5.13}$$

## 5.5 Neural Network Structure and Optimization

We optimize Equations 5.9 and 5.13 using convolutional neural networks, as they have been demonstrated to successfully encode 2D spatial relationships. The input to each neural network is the overhead image, $M_p$, and the output of each network is a discretized map that encodes the parameters of the abstract cost and traversability functions. The optimization in Equation 5.9 is constrained to output a valid probability of traversability by applying a sigmoid activation function to the network output, while the optimization in Equation 5.13 is constrained to output a non-negative mean traversal cost by applying a ReLU activation function to the network output. Both networks use an autoencoder architecture, and are optimized via back-propagation and stochastic gradient descent (SGD) with mini-batches.

## 5.6 Hierarchical Planning using Informed Abstractions

Finally, we discuss the hierarchical search algorithm used to generate solutions to the optimization in Equation 2.3. At a high level, the algorithm has two stages.

First, an uncertainty-aware discrete planner is used to find a low cost solution $\mathbf{p}$ to the optimization in Equation 2.6 using the perceptually informed abstract actions, cost function, and traversability function. Once a low cost solution is found, a high-resolution discrete planner is used to determine the lowest cost primitive refinement of $\mathbf{p}$ in the geometric map (i.e., this finds a solution to a version of Equation 2.3 that is constrained to be a refinement of $\mathbf{p}$). The true cost of the primitive refinement is recorded in the abstract planner, and search continues until it is unlikely that a lower cost plan will be found, or timeout occurs. Pseudocode for the algorithm is presented in Algorithm 1.

### 5.6.1 Uncertainty-Aware Planning using Cost Thresholds

To take advantage of the learned cost function, which models the costs of abstract actions as samples from probability distributions, we need to define an abstract planner capable of reasoning over actions with uncertain costs. In this work, we use *thresholds* to represent the lowest likely cost of an abstract action and the highest likely cost of an abstract action. Specifically, we define lower and upper thresholds for the costs of abstract actions as the lower and upper critical values of the corresponding Gaussian distributions. Formally, we define the *lower threshold* of the cost of an abstract action,

$$\mathbf{c}^{lt}(\mathbf{a}_i; \gamma) = \mu_i - \gamma\sigma, \tag{5.14}$$

where $\gamma$ is a hand-tuned parameter. Intuitively, when $\gamma$ is large, the planner considers a wider range of possible action costs, and it is less likely that the true cost of a plan will fall below the predicted lower cost threshold (i.e., the planner is *risk-averse*). However, risk-averse thresholds cause the planner to search over a larger number of plans, and are consequentially less efficient. Conversely, when $\gamma$ is small, the planner considers a smaller number of possible action costs, and can be seen as more risky, but more efficient. Similarly, we define the *upper threshold* of the cost of an abstract action,

$$\mathbf{c}^{ut}(\mathbf{a}_i; \gamma) = \mu_i + \gamma\sigma. \tag{5.15}$$

In a slight abuse of notation, we overload the threshold functions to simplify the notation for the thresholds of an abstract plan: $\mathbf{c}^{lt}(\mathbf{p}) = \sum_{t=0}^{|\mathbf{p}|-1} \mathbf{c}^{lt}(\mathbf{a}_t)$ and $\mathbf{c}^{ut}(\mathbf{p}) = \sum_{t=0}^{|\mathbf{p}|-1} \mathbf{c}^{ut}(\mathbf{a}_t)$. Upper and lower thresholds are used throughout Algorithm 1 to prune plans that are unlikely to have low cost primitive refinements.

### 5.6.2 Uncertainty-Aware Planning using Traversability Thresholds

We also use thresholds to take advantage of the learned traversability function, which models the traversability of abstract actions as Bernoulli distributions. Specifically, we define a traversability threshold $\eta$, which partitions the action space into traversable and untraversable actions based on the Bernoulli parameter of each abstract action,

$$\mathbf{f}^{t}(\mathbf{a}_i; \eta) = \begin{cases} 1 & \text{if } \pi_i > \eta \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

Intuitively, low traversability thresholds assume that all actions are more likely to be traversable, and can be seen as more risk-averse, as it is less likely to prune a plan that is traversable. However, decreasing the threshold requires the planner to search over a larger number of plans that are likely to be untraversable, which is less efficient. Conversely, high traversability thresholds assume that all actions are less likely to be traversable, and can be seen as more risky, but more efficient. In practice, we begin planning using a higher traversability threshold, and replan using a reduced threshold if no valid plan is found in the environment.

### 5.6.3 The Algorithm

The algorithm is called using the ForwardSearch function, which takes an initial state, goal region, overhead image, dense geometric map, timeout parameter, and threshold parameters as input. The algorithm initializes a planning queue, a primitive plans queue, and an uncertainty-aware visited list, and places the start node on the

planning queue and in the visited list (lines 2-4). Then, search begins. While the queue is not empty, the plan with the lowest lower threshold $\mathbf{c}^{lb}(\mathbf{p};\gamma)$ is selected for expansion (lines 6, 8). If the cost of the current best primitive plan $p^*$ is lower than the lower threshold of $\mathbf{p}$, the current plan being expanded, it is unlikely that a plan on the queue will have a lower cost plan refinement than the current best primitive plan. The likely lowest cost plan has been found, and the algorithm terminates (lines 10-11). If the final state in the current plan is the goal state, a valid abstract plan has been found, and the lowest cost primitive refinement of the abstract plan is calculated and added to the set of existing primitive plans (lines 12-14, 24-29). Otherwise, the existing plan is expanded (lines 15, 30-36). Abstract actions are selected for expansion if they are traversable (line 34) and can be part of a lower cost plan than existing plans (lines 17-18). Finally, if the queue is empty or the planner exceeds the maximum number of allowable iterations, the planer exits the planning loop. Notably, the timeout condition can occur even when a valid primitive plan has been found; this occurs when there is high uncertainty in the costs of abstract plans, so no primitive plan is identified as being sufficiently likely to be the lowest cost plan. In this situation, we return the lowest cost primitive plan found to date (lines 21-23).

**Algorithm 1** Hierarchical Forward Search using Perceptually Informed Abstractions

1: **function** ForwardSearch($x_s$, $x_g$, $M_p$, $M_g$, max-iters)
2: $Q \leftarrow \{x_s\}$
3: $plans \leftarrow \emptyset$
4: $visited \leftarrow (\{x_s\}, 0, 0)$
5: $i \leftarrow 0$
6: **while** $|Q| > 0$ and $i <$max-iters **do**
7: $i \leftarrow i + 1$
8: $\mathbf{p} \leftarrow \arg\min\{\mathbf{c}^{lb}(\mathbf{p}; \gamma) : \mathbf{p} \in Q\}$
9: $p^* \leftarrow \arg\min\{c(p) : p \in plans\}$
10: **if** $c(p^*) \leq \mathbf{c}^{lb}(\mathbf{p}; \gamma)$ **then**
11: **return** $p^*$
12: **if** $\mathbf{p}.last() = x_g$ **then**
13: $p_{ref} \leftarrow$ Refine($x_s$, $x_g$, $M_g$, $\mathbf{p}$)
14: $plans \leftarrow plans \cup p_{ref}$
15: **for all** $n \in$ Expand($\mathbf{p}, \eta$) **do**
16: $\mathbf{p}' = \mathbf{p} \circ n$
17: $p_n \leftarrow$ MinimumUpperBoundVisitation($n$)
18: **if** $\neg visited(n)$ or $\mathbf{c}^{lb}(p'; \gamma) < \mathbf{c}^{ub}(p_n; \gamma)$ **then**
19: $Q \leftarrow \{p'\}$
20: $visited \leftarrow (n, \mathbf{c}^{lb}(p'; \gamma), \mathbf{c}^{ub}(p'; \gamma))$
21: **if** $|plans| > 0$ **then**
22: $p \leftarrow \arg\min\{c(p) : p \in plans\}$
23: **return** p
**return** $\emptyset$
24: **function** Refine($x_s$, $x_g$, $M_g$, $\mathbf{p}$)
25: $M_g^{\mathbf{p}} \leftarrow$ RestrictMapToRegionsInAbstractPlan($M_g, \mathbf{p}$)
26: $p \leftarrow$ PrimitiveForwardSearch ($x_s$, $x_g$, $M_g^{\mathbf{p}}$)
27: **if** $\neg p = \emptyset$ **then**
28: UpdateBounds($p$)
29: **return** $p$
**return** $\emptyset$
30: **function** Expand($\mathbf{p}$; $\eta$)
31: $neighbors \leftarrow \emptyset$
32: $candidate\_neighbors \leftarrow$ AbstractNeighbors($\mathbf{p}$)
33: **for** $n \in candidate\_neighbors$ **do**
34: **if** $\mathbf{f}^t(n; \eta)$ **then**
35: UpdateBounds($n$)
36: $neighbors \leftarrow neighbors \cup n$
**return** $neighbors$

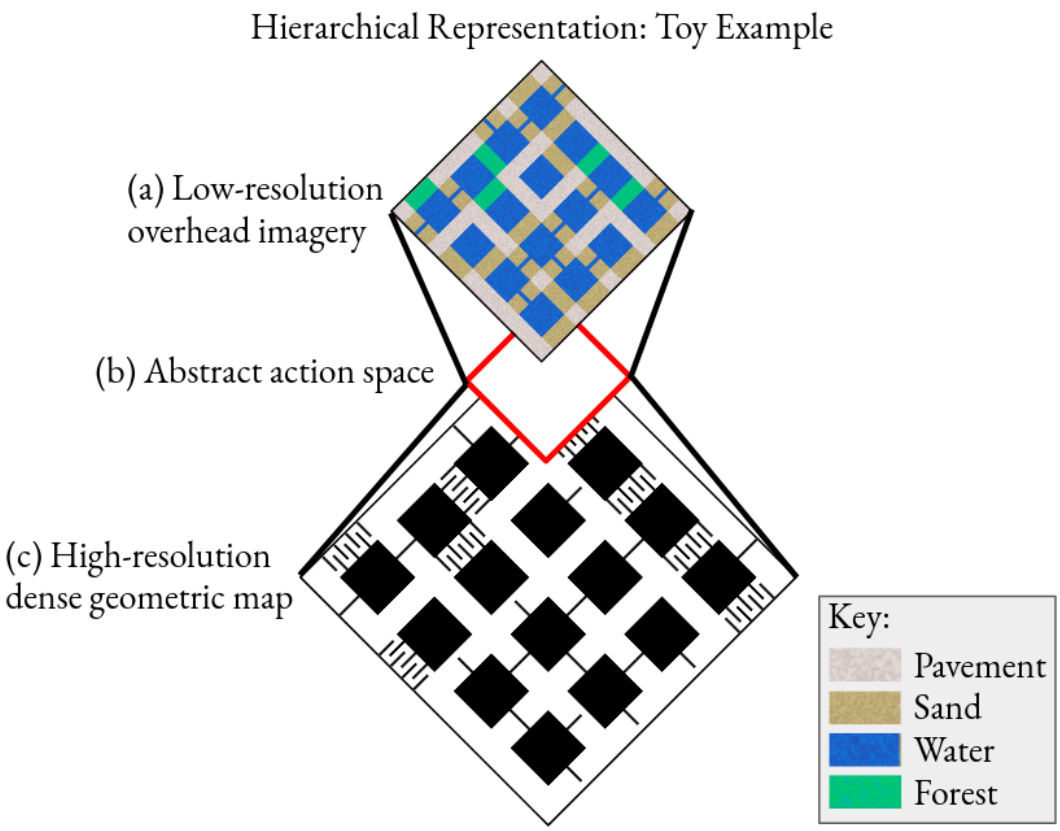


Figure 5-2: **Overview of hierarchical planning in the toy example environment.** Our approach uses overhead imagery (a) to approximate the costs of abstract actions in an abstract action space (b). Once abstract plans are found in abstract action space, they are used to guide the search for primitive plans in the dense geometric map (c).

## 5.7 Preliminary Experiments

In this section, we present preliminary evaluations of the Perceptually Informed Abstractions approach in a toy forest environment. We learned a traversability function $f(\mathbf{a})$ using the approach in Section 5.4.1 and evaluated the function's classification accuracy for different thresholds $\eta$. We also analyzed the performance of the hierarchical planning approach using ground truth abstract traversability and cost functions as compared to a baseline high-resolution A*-based planner, and discuss the challenges of using uncertain abstract actions for hierarchical planning.

### 5.7.1 Experimental Setup

We tested our approach in simulation in a toy forest environment. We procedurally generated a dataset of 300 toy outdoor maps, which were split into a training set of 200 environments and a testing set of 100 environments. An example toy environment is shown in Figure 5-2. Each environment contained two modes of information – a high-resolution dense geometric map (832 by 832 pixels, e.g., Figure 5-2c) and a low-resolution overhead image (208 by 208 pixels, e.g., Figure 5-2b). We assumed that the resolution of the abstract action space was 1/64 the resolution of the dense geometric map, and 1/16 the resolution of the overhead image (Figure 5-2a). The high-resolution dense geometric map was a standard occupancy map with different geometries to represent traversals in different terrains. For example, a maze geometry was used to approximate the agent's navigation behavior when traversing through trees in a forest or around mud in select sandy areas. The simulated overhead image was comprised of four noisy terrain classes: water, forest, sand, and pavement. We assumed that water was untraversable in the dense geometric map, and that all other terrain types were traversable, although terrain traversal costs were dependent on the geometric structure of the dense geometric map. We assumed that the agent was a holonomic ground vehicle with no sensor or pose estimation noise, and that the objective of the agent was to find minimum length primitive plans while avoiding obstacles (i.e., using soft costs).

To generate the dataset $\mathcal{D}$, we simulated 3000 optimal planning trials in the 300 training and testing environments by randomly selecting starts and goals and running A* in the dense geometric map to generate optimal primitive trajectories[4]. We recorded tuples of high-resolution geometric maps, low-resolution overhead images, and optimal trajectories, then post-processed the dataset to recover cost and traversability labels for different abstract actions as described in Section 5.4.

[4]We removed trials where no valid primitive plan was found.

### 5.7.2 Learned Traversability

First, we demonstrated the effectiveness of learning a traversability function for hierarchical navigation. We optimized an autoencoder-based model to maximize the objective in Equation 5.9 over the training dataset using the procedure described in Section 5.5. Then, we compared the classification accuracy of the learned traversability function under different thresholds to the ground truth traversability labels over the test dataset, which comprised of 67,600 individual abstract actions and traversability labels in 100 distinct environments. Plots of the overall model accuracy, as well as true positive and negative rates of the model, are shown in Figure 5-3. The model achieves a maximum accuracy of 99.0% with a traversability threshold of 0.48, and achieves similar accuracies for a wide range of thresholds, indicating that the approach is robust to the threshold parameter (Figure 5-3a). Additionally, we consider the classification accuracy of true positive and true negative samples (Figure 5-3b), and demonstrate that the network is capable of predicting the traversability values of both traversable and untraversable regions in the environment. These results indicate that low-resolution overhead imagery can be used to predict a traversability function over abstract actions in the toy environment.

### 5.7.3 Hierarchical Planning using Ground Truth Abstract Functions

Second, we analyzed the hierarchical planning algorithm, independent of the learned cost and traversability functions, by simulating hierarchical planning using ground truth abstract function values, which could be computed or approximated in the toy environment[5]. We randomly selected start and goal locations in the 100 test environments, and simulated hierarchical planning until a valid plan was found and returned, or until the timeout condition of 10,000 hierarchical iterations was met[6].

[5]For this experiment, we calculated ground truth function values using the testing dataset, but expect that similar ground truth values would be obtained from the training dataset. We assumed a cost function standard deviation of 1 for all trials, $\sigma = 1$.

[6]We guaranteed that a valid primitive plan existed for each of the trials.

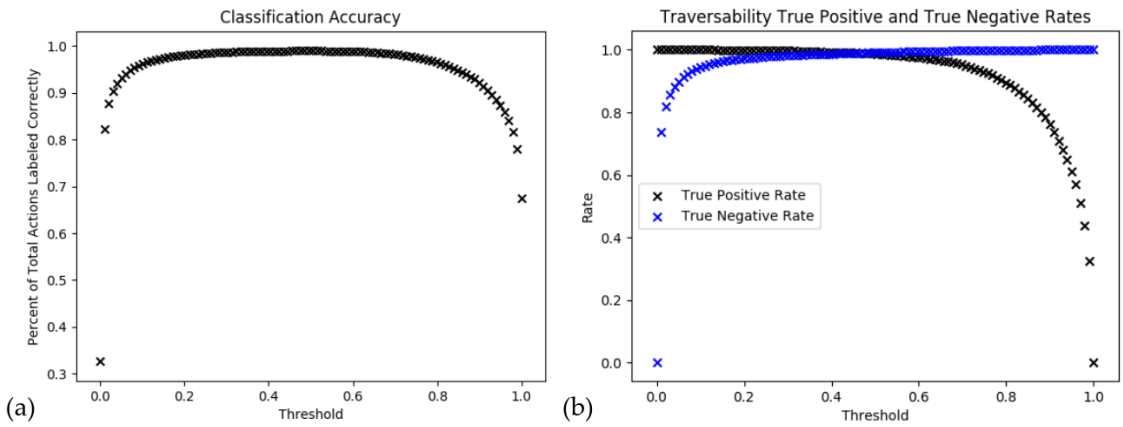


Figure 5-3: **Traversability classification accuracy.** We analyzed the (a) classification accuracy and (b) true positive and true negative rates for the learned traversability function for different thresholds $\eta$ over 67,600 individual abstract actions in the test environment. The network predicts traversability with high accuracy for a wide range of thresholds, indicating that our approach is capable of learning a robust abstract traversability function using low-resolution overhead imagery in the toy environment.

We compared the navigation outcomes of the hierarchical planner to the outcomes of a baseline planner, which used the A* algorithm to search over the dense geometric map of the toy environment. We evaluated the number of planner nodes expanded, the planner wallclock time, and the costs of plans generated by the two approaches.

To begin, we compared the navigation outcomes of the two planners when the hierarchical planner assumed no cost function uncertainty (i.e., $\gamma = 0$). We present the results of the experiment in Figure 5-4, and use linear regressors with zero intercept to generate aggregate statistics that compare the test metrics for the planners. First, we consider the total number of plans expanded by the planner. The hierarchical planning approach expanded 64% fewer plans than the A* algorithm, as determined by linear regression. However, the hierarchical planning algorithm involved more overhead computation per plan expanded than the A* algorithm, resulting in significantly longer planning times (approximately 4.59 times longer, according to linear regression). Finally, we compared the relative costs of plans generated by the two approaches. The cost of the baseline A* plan is resolution optimal in the dense geometric map, and is therefore the lowest cost plan that can be found in the environment, given the dense geometric model. Because the hierarchical planner refines

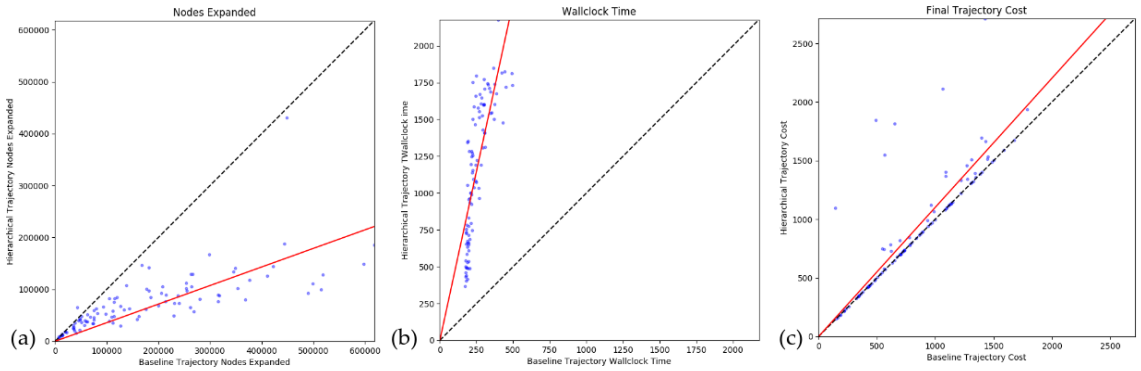


Figure 5-4: **Comparison of plans generated by the hierarchical and baseline planners with no abstract cost uncertainty.** Each blue circle represents one planning trial. The red line is a linear regressor, and the black dashed line is an equal cost reference. While the hierarchical planner explored fewer plans than the baseline planner (a), the baseline planner generated plans more quickly, partially due to hierarchical planning overhead (b). However, despite considering a smaller number of planning nodes, the hierarchical planner found plans that were only 10% less efficient than the optimal plan costs, as determined by linear regression (c), indicating that overhead imagery can be used to inform intelligent navigation behaviors.

abstract plans using A* over a small subset of the dense geometric map, the goal of the hierarchical planner is to efficiently recover a plan that is no more expensive than the plan recovered by the optimal baseline. We demonstrated that, despite expanding fewer possible plans than the A* algorithm, our hierarchical planner recovered the optimal plan cost in 62% of the trials, and resulted in plans that were only 10% more expensive on average than the optimal A* plan, as determined by linear regression. This result indicates that overhead images provide useful navigation cues for the planner, as the hierarchical algorithm was able to guide efficient planning in the dense geometric map using only low-resolution overhead image data. In future work, we intend to optimize the hierarchical planner implementation, and expect to be able to reduce hierarchical planning wallclock time.

However, we would also like to be able to explicitly consider planning uncertainty in the hierarchical planner. Reasoning about planning uncertainty will be especially important when transitioning our approach to real-world environments, where overhead imagery is more noisy, and where there may be more variation in plan costs within terrain classes in the environment. To test planning with uncertain cost func-

| **Cost Uncertainty Scale Factor, $\gamma$** | **0.0** | **2.0** | **5.0** | **10.0** | **15.0** |
|---|---|---|---|---|---|
| $\bar{\boldsymbol{N}}$ | 0.36 | 0.93 | 14.29 | 23.78 | 31.03 |
| $\boldsymbol{R^2}$ Score, $\bar{\boldsymbol{N}}$ | 0.47 | 0.11 | 0.16 | 0.015 | 0.10 |
| $\bar{\boldsymbol{T}}$ | 4.59 | 6.23 | 57.50 | 298.75 | 409.73 |
| $\boldsymbol{R^2}$ Score, $\bar{\boldsymbol{T}}$ | 0.62 | 0.19 | 0.10 | 0.19 | 0.28 |
| $\bar{\boldsymbol{C}}$ | 1.10 | 1.11 | 1.09 | 1.08 | 1.10 |
| $\boldsymbol{R^2}$ Score, $\bar{\boldsymbol{C}}$ | 0.69 | 0.72 | 0.71 | 0.83 | 0.69 |
| $\|\|\boldsymbol{D}\|\|$ | 100 | 100 | 100 | 100 | 100 |
| $\|\|\boldsymbol{D_h}\|\|$ | 100 | 100 | 99 | 85 | 69 |
| $\|\|\boldsymbol{D_{to}}\|\|$ | 0 | 0 | 1 | 37 | 60 |

Table 5.1: **A comparison of hierarchical and baseline plan metrics for different cost function uncertainties.** We compared planning metrics for the hierarchical and baseline planners when both planners succeeded, broken out by cost function scaling parameter $\gamma$. $\bar{\boldsymbol{N}}$, $\bar{\boldsymbol{T}}$, and $\bar{\boldsymbol{C}}$ are the slopes of the linear fit regressors for the number of nodes expanded, planner wallclock time, and resulting plan cost metrics, respectively. $||\boldsymbol{D}||$ is the total number of planning trials considered, and $||\boldsymbol{D_h}||$ is the number of trials where the hierarchical planner found a plan (the baseline planner found a plan for every trial). Finally, $||\boldsymbol{D_{to}}||$ is the number of trials where the hierarchical planner timed out. We remind the reader that the hierarchical planner can time out and find a plan, as the hierarchical planner returns a plan, if one has been found, after timing out. While the hierarchical planner successfully reduces the number of nodes expanded without significantly compromising plan quality for low values of $\gamma$, the planner is expensive and inefficient for large values of $\gamma$. We discuss mitigations for these inefficiencies in the text.

tions, we compare the performance of the hierarchical and baseline planners for various values of $\gamma$. Complete results of the experiment are given in Table 5.7.3.

In general, we demonstrate that plan cost uncertainty grows quickly in the toy environment, and for high values of $\gamma$, the planner reverts to exhaustive search behaviors. In some cases, the hierarchical algorithm inefficiently traverses positive cost cycles, as loose upper and lower bounds on plan costs can make it challenging to prune plans with high cost uncertainty[7]. In other cases, the planner finds a valid plan relatively quickly, but is unable to ascertain that the plan is likely to be lower cost than other plans in the environment due to high cost uncertainty. This leads to inefficient planning behavior, as the planner must refine many uncertain plans before determining that an existing solution is the likely lowest cost plan. We visually com-

[7]We note that we cannot simply eliminate cyclic hierarchical plans in the environment, as optimal primitive behavior can be cyclic in the abstract space.

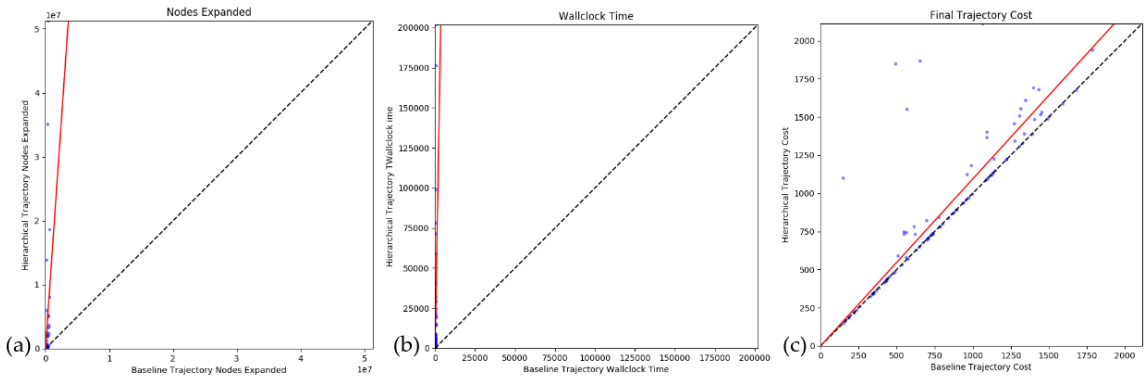


Figure 5-5: **Comparison of plans generated by the hierarchical and baseline planners with abstract cost uncertainty.** Each blue circle represents one planning trial. The red line is a linear regressor, and the black dashed line is an equal cost reference. The cost uncertainty factor used in this example was $\gamma = 5$. When using a larger cost uncertainty factor, the hierarchical planner was unable to use the abstract cost function to prune plans that were unlikely to be low cost, and performed poorly relative to the baseline for all calculated metrics. In this section, we discuss methods to mitigate myopic planning behavior caused by cost uncertainty.

pare planner metrics for the hierarchical planner with $\gamma = 5$ and the baseline planner in Figure 5-5.

While the existing approach is inefficient for abstract cost functions with high cost uncertainty, the challenges presented here are well-known in the hierarchical planning under uncertainty literature, and a number of approaches have been proposed to mitigate their effects. For example, Vega-Brown and Roy [90] proposed avoiding positive cost cycles in hierarchical planners by *deferring* cyclic plans until parent plan nodes had been further refined, ensuring that no plan contained two identical actions at the same planning resolution. While this approach relied on the ability to individually refine abstract actions, which is not currently possible in our approach, it is possible that a similar approach, which defers the consideration of cyclic plans until other non-cyclic plans have been expanded, could improve hierarchical planning outcomes in our approach on average. Additionally, a number of authors have proposed strategies that avoid the over-exploration of plans with similar costs by biasing search to plans that have low cost-to-go values, and terminating when low cost, but not necessarily optimal, plans are found. This idea was introduced in Weighted A*

[71] and applied to hierarchical planners that guarantee primitive plan optimality in [90]. In future work, we would like to explore the application of these mitigations to our hierarchical planning approach.

# Chapter 6

# Conclusion

This thesis presented two methods that used learning to incorporate perceptual information into classical planning techniques for informed robot navigation in structured environments. While existing informed planning techniques relied on dense geometric information to plan intelligently in structured environments, we demonstrated that perceptual information can be used to inform efficient robot navigation in structured unknown and long length scale environments. By using learned functions informed by optimal trajectories to incorporate perceptual information into planning, we avoided explicitly defining the ways in which perceptual information informs navigation.

In Chapter 3, we presented Learned Sampling Distributions, a novel method for learning a sampling distribution based on local hybrid geometric and object-level maps to inform a sampling-based motion planner for navigation in unknown environments. Our approach used example optimal trajectories to learn a probability distribution that places high probability in regions of the environment that are likely to be on optimal paths to the goal, like hallways and doorways in an office environment. In Chapter 4, we demonstrated up to a 2.7x increase in the probability of finding a plan using our learned approach as compared to an uninformed baseline planner, and up to a 16% reduction in plan cost when using our learned approach as compared to an uninformed planner. We also demonstrated promising results on a 1/10th scale RC car platform navigating online in a building at MIT.

In Chapter 5, we presented Perceptually Informed Abstractions, a novel method

for risk-aware planning at long length scales that learns the properties of abstract actions for use in a risk-aware hierarchical discrete planner. Our approach proposed using ground-truth traversability annotations and optimal trajectories to learn the traversability and cost distributions of abstract actions, respectively, conditioned on low-resolution overhead images. We proposed combining our learned abstractions with a risk-aware hierarchical planner to generate perceptually-informed navigation strategies. We also presented preliminary results for learning a traversability function from overhead images, demonstrated the hierarchical planning approach using ground truth cost and traversability functions, and discussed the challenges of hierarchical planning with uncertain cost functions.

There are a number of interesting extensions to the work presented in this thesis. First, the Perceptually Informed Abstractions work is in an early phase of development, and we would like to demonstrate a more complete set of results, including learning cost functions and demonstrating improved planning performance using the hierarchical planner. Additionally, we would like to demonstrate Perceptually Informed Abstractions in a more realistic environment, such as the university floorplans environment. Additional experimentation is required to demonstrate the robustness of the approach to diverse navigation scenarios in real-world environments.

Another interesting extension to the approaches presented in this thesis would be the use of different cost functions for planning. While the experiments in this thesis only consider a total path length objective, neither of the approaches are limited to using a path length objective. By employing a different cost function to score optimal trajectories, the approaches may be extended to demonstrating more complex behaviors. For example, a visibility-aware cost function or communication-aware cost function could be used to encourage stealthy navigation for surveillance applications.

Finally, the implementations in this thesis are limited to navigation problems that occur in low-dimensional state spaces; this is because we chose to represent learned functions as discrete functions defined over the robot's configuration space. This representation was chosen to exploit the ability of convolutional neural networks to represent 2D spatial relationships. However, the methods proposed are not tied to

a specific function representation, provided that the representation can be parameterized by learnable weights. For example, one interesting extension might be to use the proposed optimization schemes to learn the statistics of a multivariate Gaussian mixture model to represent the learned functions which may be more computationally tractable to define for a high-dimensional state space.

# Bibliography


[1] Intel© NUC Kit NUC8i7BEH, 2020. URL `https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc8i7beh.html`.

[2] Intel© RealSense™ Depth Camera D435i, 2020. URL `https://www.intelrealsense.com/depth-camera-d435i/`.

[3] Intel© RealSense™ Tracking Camera T265, 2020. URL `https://www.intelrealsense.com/tracking-camera-t265/`.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[5] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, 1998.

[6] I. Baldwin and P. Newman. Non-parametric learning for natural plan generation. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4311–4317. IEEE, 2010.

[7] I. Baldwin and P. Newman. Teaching a randomized planner to plan with semantic fields. *TAROS 2010*, page 20, 2010.

[8] S. Behnke. Local multiresolution path planning. In *Robot Soccer World Cup*, pages 332–343. Springer, 2003.

[9] C. H. Bennett. Efficient estimation of free energy differences from monte carlo data. *Journal of Computational Physics*, 22(2):245–268, 1976.

[10] M. Blaha, C. Vogel, A. Richard, J. D. Wegner, T. Pock, and K. Schindler. Large-scale semantic 3d reconstruction: an adaptive multi-resolution model for multi-class volumetric labeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3176–3184, 2016.

[11] V. Boor, M. H. Overmars, and A. F. Van Der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1018–1023. IEEE, 1999.

[12] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 3120–3125. IEEE, 2005.

[13] B. Burns and O. Brock. Toward optimal configuration space sampling. In *Robotics: Science and Systems*, pages 105–112. Cambridge, USA, 2005.

[14] J. Canny. *The complexity of robot motion planning*. MIT press, 1988.

[15] W. G. Chase. Spatial representations of taxi drivers. In *The acquisition of symbolic skills*, pages 391–405. Springer, 1983.

[16] M. Costandi. How do brain cells tell us where we're going: New findings provide a more complex profile of the brain's "internal gps". *Scientific American*. URL `https://www.scientificamerican.com/article/how-do-brain-cells-tell-us-where-were-going/`.

[17] E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[18] S. L. Epstein. Navigation, cognitive spatial models, and the mind. In *2017 AAAI Fall Symposium Series*, 2017.

[19] S. L. Epstein, A. Aroor, M. Evanusa, E. I. Sklar, and S. Parsons. Learning spatial models for navigation. In *International Conference on Spatial Information Theory*, pages 403–425. Springer, 2015.

[20] M. Everett, J. Miller, and J. P. How. Planning beyond the sensing horizon using a learned context. *arXiv preprint arXiv:1908.09171*, 2019.

[21] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[22] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[23] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[24] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Batch informed trees (bit*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5):543–567, 2020.

[25] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[26] A. V. Goldberg. Point-to-point shortest path algorithms with preprocessing. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 88–102. Springer, 2007.

[27] K. Grant and D. Mould. Lpi: Approximating shortest paths using landmarks. In *Workshop on Artificial Intelligence in Games*, page 45, 2008.

[28] L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*, volume 1, pages 254–259, 1999.

[29] P. E. Hart, N. J. Nilsson, and R. Bertram. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, pages 100–1072. IEEE, 1968.

[30] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, et al. Bop: Benchmark for 6d object pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[31] T. Hodan, D. Barath, and J. Matas. Epos: Estimating 6d pose of objects with symmetries. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[32] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. doi: 10.1007/s10514-012-9321-0. URL `http://octomap.github.com`. Software available at `http://octomap.github.com`.

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[34] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *2003 IEEE international conference on robotics and automation (cat. no. 03CH37422)*, volume 3, pages 4420–4426. IEEE, 2003.

[35] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[36] B. Ichter, E. Schmerling, T.-W. E. Lee, and A. Faust. Learned critical probabilistic roadmaps for robotic motion planning. pages 9535–9541, 2020.

[37] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan. Rrt-smart: Rapid convergence implementation of rrt towards optimal solution. In *2012 IEEE International Conference on Mechatronics and Automation*, pages 1651–1656. IEEE, 2012.

[38] L. Jaillet, J. Cortés, and T. Siméon. Transition-based rrt for path planning in continuous cost spaces. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2145–2150. IEEE, 2008.

[39] L. Janson, E. Schmerling, A. Clark, and M. Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[40] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[41] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[42] L. E. Kavraki, 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.

[43] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[44] T. Klamt and S. Behnke. Towards learning abstract representations for locomotion planning in high-dimensional state spaces. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 922–928. IEEE, 2019.

[45] R. A. Knepper and M. T. Mason. Real-time informed path sampling for motion planning search. *The International Journal of Robotics Research*, 31(11):1231–1250, 2012.

[46] G. Konidaris, 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.

[47] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, et al. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982*, 2018.

[48] D. T. Larsson, D. Maity, and P. Tsiotras. Q-search trees: An information-theoretic approach towards hierarchical abstractions for agents with computational limitations. *arXiv preprint arXiv:1910.00063*, 2019.

[49] D. T. Larsson, D. Maity, and P. Tsiotras. An information-theoretic approach for path planning in agents with computational constraints. *arXiv preprint arXiv:2005.09611*, 2020.

[50] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[51] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[52] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[53] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[54] D. J. C. MacKay. *Information Theory, Inference Learning Algorithms*. Cambridge University Press, USA, 2002. ISBN 0521642981.

[55] B. Marthi, S. J. Russell, and J. A. Wolfe. Angelic semantics for high-level actions. 2007.

[56] B. Marthi, S. Russell, J. Wolfe, et al. Angelic hierarchical planning: Optimal and online algorithms (revised). 2009.

[57] M. C. Martin and H. P. Moravec. Robot evidence grids. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Robotics Inst, 1996.

[58] Microsoft. *Bing Maps*, 2020 (accessed August 8, 2020). URL `https://www.bing.com/maps`.

[59] D. Molina, K. Kumar, and S. Srivastava. Learn and link: Learning critical regions for efficient planning. pages 10605–10611, 2020.

[60] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33 (5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.

[61] L. Murphy and P. Newman. Planning most-likely paths from overhead imagery. In *2010 IEEE International Conference on Robotics and Automation*, pages 3059–3064. IEEE, 2010.

[62] L. Murphy and P. Newman. Risky planning on probabilistic costmaps for path planning in outdoor environments. *IEEE Transactions on Robotics*, 29(2):445–457, 2012.

[63] L. Nicholson, M. Milford, and N. Sünderhauf. Quadricslam: Dual quadrics from object detections as landmarks in object-oriented slam. *IEEE Robotics and Automation Letters*, 4(1):1–8, 2019.

[64] M. Nieuwenhuisen, D. Droeschel, M. Beul, and S. Behnke. Obstacle detection and navigation planning for autonomous micro aerial vehicles. In *2014 international conference on unmanned aircraft systems (ICUAS)*, pages 1040–1047. IEEE, 2014.

[65] K. Ok, K. Liu, K. Frey, J. P. How, and N. Roy. Robust object-based slam for high-speed autonomous navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 669–675. IEEE, 2019.

[66] C. Packham. Study demonstrates how humans navigate through doorways and not into walls, 2017. URL `https://medicalxpress.com/news/2017-04-humans-doorways-walls.html`.

[67] D. K. Pai and L.-M. Reissell. Multiresolution rough terrain motion planning. *IEEE Transactions on robotics and automation*, 14(1):19–33, 1998.

[68] J. Pearl. Intelligent search strategies for computer problem solving. *Addision Wesley*, 1984.

[69] J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4):392–399, 1982.

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

[71] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[72] A. Pronobis, F. Riccio, and R. P. Rao. Deep spatial affordance hierarchy: Spatial knowledge representation for planning in large-scale environments. In *ICAPS 2017 Workshop on Planning and Robotics*, June 2017.

[73] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427. IEEE, 1979.

[74] A. Rosinol, M. Abate, Y. Chang, and L. Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. *arXiv preprint arXiv:1910.02490*, 2019.

[75] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, 1989.

[76] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1352–1359, 2013.

[77] A. Shkolnik and R. Tedrake. Sample-based planning with volumes in configuration space. *arXiv preprint arXiv:1109.3145*, 2011.

[78] D. Silver, B. Sofman, N. Vandapel, J. A. Bagnell, and A. Stentz. Experimental analysis of overhead data processing to support long range navigation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2443–2450. IEEE, 2006.

[79] D. Silver, J. A. Bagnell, and A. Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010.

[80] D. Silver, J. A. Bagnell, and A. Stentz. Active learning from demonstration for robust autonomous navigation. In *2012 IEEE International Conference on Robotics and Automation*, pages 200–207. IEEE, 2012.

[81] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[82] G. J. Stein, C. Bradley, and N. Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In *Conference on Robot Learning*, pages 213–222, 2018.

[83] G. J. Stein, C. Bradley, V. Preston, and N. Roy. Enabling topological planning with monocular vision. In *2020 International Conference on Robotics and Automation (ICRA)*, pages 1667–1673. IEEE, 2020.

[84] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `https://ompl.kavrakilab.org`.

[85] N. Sünderhauf. Where are the keys?–learning object-centric navigation policies on semantic maps with graph convolutional networks. *arXiv preprint arXiv:1909.07376*, 2019.

[86] O. Takahashi and R. J. Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989.

[87] C. Thorpe and L. Matthies. Path relaxation: Path planning for a mobile robot. In *OCEANS 1984*, pages 576–581. IEEE, 1984.

[88] C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1178–1183. IEEE, 2003.

[89] J. P. Van den Berg and M. H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. *The International Journal of Robotics Research*, 24(12):1055–1071, 2005.

[90] W. Vega-Brown and N. Roy. Admissible abstractions for near-optimal task and motion planning. In *International Joint Conference on Artificial Intelligence*, Stockholm, 2018.

[91] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng. Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 2020.

[92] M. Wigness, J. G. Rogers, and L. E. Navarro-Serment. Robot navigation from human demonstration: Learning control behaviors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1150–1157. IEEE, 2018.

[93] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[94] M. Wulfmeier, D. Z. Wang, and I. Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2089–2095. IEEE, 2016.

[95] H.-Y. Yeh, S. Thomas, D. Eppstein, and N. M. Amato. Uobprm: A uniformly distributed obstacle-based prm. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2655–2662. IEEE, 2012.

[96] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 3856–3861. IEEE, 2005.

[97] C. Zhang, J. Huh, and D. D. Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661. IEEE, 2018.

[98] K. Zheng, A. Pronobis, and R. P. Rao. Learning graph-structured sum-product networks for probabilistic semantic maps. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[99] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[100] M. Zucker, J. Kuffner, and J. A. Bagnell. Adaptive workspace biasing for sampling-based planners. In *2008 IEEE International Conference on Robotics and Automation*, pages 3757–3762. IEEE, 2008.

[101] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.