Navigation of Unknown Environments Using High-Level Actions

by

Christopher Powell Bradley

B.S., California Institute of Technology (2017)

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 2019

Author . . . .
Department of Aeronautics and Astronautics
August 22, 2019

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

Accepted by . . . .
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Navigation of Unknown Environments Using High-Level Actions

by

Christopher Powell Bradley

Submitted to the Department of Aeronautics and Astronautics
on August 22, 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Goal-oriented, autonomous navigation through previously unexplored environments presents challenges to a robot on a number of different fronts. First, the robot must construct a representation of its environment that enables it to reason about entering and exploring unknown regions, while still allowing it to backtrack through previously explored space. Additionally, the robot must be able evaluate the expected cost of plans through unobserved space to reach its objective efficiently.

This thesis presents work that addresses each of these challenges with respect to a mobile robot. The Learned Subgoal Planner provides an abstraction for planning using high-level actions to reduce the complexity of the planning problem. Using learning to estimate the cost of different actions, a 21% improvement in terms of distance traveled versus a baseline was shown in a simulated environment replicating real-world floor plans. A second contribution is a novel mapping paradigm which represents the world with a graph of actions build from monocular visual input. To construct this map, a convolutional network is used to detect high-level actions from vision. The map is shown to be robust to noise, with particular attention paid to the problem of associating detected actions from frame to frame using a learned association metric. Preliminary results show this metric is an improvement compared to a baseline.

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

Acknowledgments

I have been privileged throughout my life to have been surrounded by people who have given me the opportunity to a receive world-class education. First and foremost, I would like to thank my parents for supporting my studies, and encouraging me to pursue my interests, regardless of application or field. Thank you to Nick Roy for allowing me to join the Robust Robotics Group, even with the knowledge that I was new to the field of robotics and would need a bit of time and support to get up to speed. Thank you to Greg Stein for welcoming collaboration, and for helping to shape how I think about conducting and presenting research. Thanks to Guillaume Blanquart, Joel Burdick, and Beverley McKeon at Caltech for guidance in helping me find my way to graduate school. Finally, thanks to everyone in RRG for sharing your inspirational research, passing along institutional knowledge, and making the lab a fun and welcoming place to go to work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Navigating an unknown environment to find an unseen goal is a ubiquitous problem in robotics, and can be broken down into two main tasks. First, the robot must be able to reason about what path to follow through unexplored space that will most efficiently lead it to the goal. Second, the robot will need to build a map as it travels such that the agent is able to localize itself and make plans that avoid local minima (dead-ends) in space that has already been explored. Research to address each of these problems is presented in this thesis.

1.1 Motivating Navigation of Unexplored Environments

The field of mobile robotics has been rapidly expanding over the past thirty years, driven in part by the myriad of possible applications for a system that is able to autonomously travel through, and accomplish tasks in, its environment. Problems as impactful as surveillance, reconnaissance, and search and rescue are familiar motivators of research in the field, but even simple, every-day tasks such as delivering coffee in an office building are potential applications of an autonomous agent [47, 9].

Independent of the larger, overall goal, for a mobile robot, a critical component of accomplishing any task is navigation. For example, if a robot is tasked with making breakfast, it first must go to the kitchen. When the map of the environment is known, it is well understood how to best plan a path to a goal in order to accomplish such a task. Dijkstra's algorithm, A*, and other search algorithms are guaranteed to return the minimum-distance path through a discretized state-space [12, 24]. Similarly, sampling-based approaches such as RRT* and PRM* are asymptotically optimal through known, continuous spaces [30]. However, when the environment is not fully known, it is no longer possible to use these well-established algorithms without reasoning about planning through unknown space.

Despite the continued advancements in other areas of the field however, there has not been much practical development in the area of planning to navigate through unknown space. Imagine a robot tasked with navigating to a goal in a large office complex with a partial map, discovering the environment as it travels. To reach its goal, the agent must plan to enter unknown parts of the map, and in so doing, reason about the cost of trajectories through regions which have not been observed. Many approaches to planning attempt to avoid the difficulties associated with reasoning about unknown space by optimistically assuming all unobserved space is free of obstacles [4, 20]. Yet if a robot were instructed to travel to a conference room on the far side of a building, for example, such an optimistic planner is likely to make globally sub-optimal decisions and guide the robot into countless offices as it navigates towards its goal as shown in Fig. 1-1. Moreover, if the task does not have a goal defined in metric space, such as the problem of search and rescue for example, then naively planning through unknown space without explicitly reasoning about where to navigate to is not a useful approach.

The central algorithm of this thesis, the Learned Subgoal Planner, seeks to solve the problem of navigating unknown environments by first decomposing the world into high-level actions, then using a data-driven approach to estimate the different costs associated with taking each action. Specifically, we associate an action with each boundary between free and unknown space and define high-level actions as traveling to the goal through each of these separate boundaries. By treating navigation as a decision between entering unknowns space through different frontiers, we greatly reduce

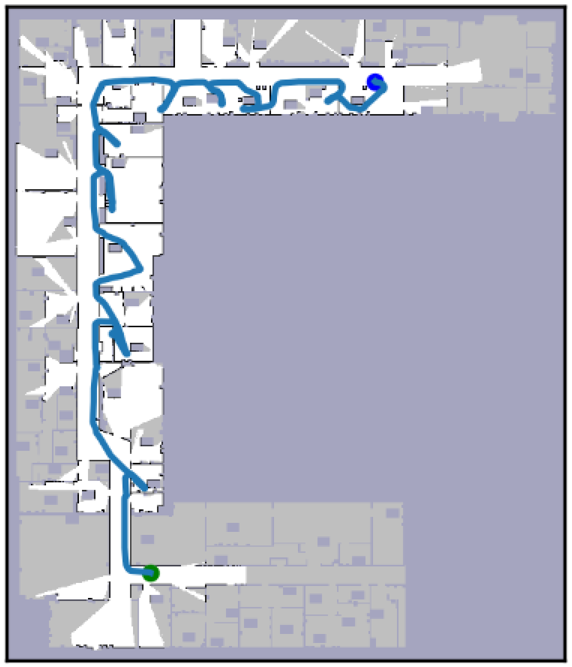

Figure 1-1: A depiction of the path a robot might follow if it were to attempt to navigate through a previously unknown building by naively assuming all unknown space is free. Here the robot begins at the blue point, and encounters many dead-ends on it's path to the green goal. An intelligent planner should realize that small offices are not likely to lead to the goal, and the robot would be better served planning through hallways.

the dimensionality of the problem, and so make estimating the cost of each action, and then finding the sequence of actions that minimizes expected cost, computationally tractable.

The set of actions defined above fully represents what a robot can do at any time, and can therefore serve as a sufficient representation for navigation. This thesis presents the notion of using a "graph of actions" as a map for navigation, which allows us to build a sparse representation of the world through vision that enables planning through known and unknown space by directly reasoning about actions. Specifically highlighted in this thesis from that work is the problem of data-association: deciding if a detected action corresponds with one already in the map, or if it represents a new action that must be added. Using the action-graph built from this method as the representation for the Learned Subgoal Planner is left for future work.

1.2 Path Planning in Different Levels of Abstraction

When an autonomous agent seeks to navigate to a specified goal, it must solve a path planning problem. This means that the robot must search for some way to safely (e.g., without colliding with any obstacles) transition from its initial configuration to the goal state, often with the objective of being optimal with respect to some cost function. An important factor in the complexity of such a problem is the level of abstraction in which it is solved. We can define an abstract representation as one that "can be constructed from a concrete [abstraction] by ignoring details and including only those aspects of primary importance [59]." In other words, a higher level abstraction is formed by discarding information that is not necessary to some task (here that task is navigation). By ignoring unnecessary information, it is possible to reduce the state space and action set of a problem, thus making it easier to solve. For example if the mobile robot tasked with navigating across a building had a mounted arm attached to it for grasping, one might choose to plan in an abstraction where the arm is assumed to be stationary, because all solutions that traverse the optimal path to the goal are of the same cost regardless of what the arm is doing.

Abstractions produce useful plans as long as the information they discard is either completely irrelevant to the task, as in the above example, or in some other way not necessary. The solution to a navigation task would, at the very lowest level, yield a sequence of motor commands that cause the robot to traverse the minimum cost path from start to goal. In practice however, long term plans are generally not produced by searching through motor commands until one sequence is found to be optimal. Instead, often times the final plan is represented by a trajectory — a sequence of valid configurations from start to goal — which some low-level controller is then responsible for producing the motor commands to follow. As long each intermediate step taken from the trajectory exists along the optimal path to the goal, and the agent is capable of optimally planning between each step in the trajectory, the ultimate path the robot follows will be the most cost-effective.

One reason path planning problems are often not solved at their lowest level of abstraction is for ease of computation. Consider, once again, a wheeled robot planning a route to some goal across an office building. One way to plan through this world would be to solve the optimal control problem using some non-linear programming solver, and produce the full sequence of commands — velocities and wheel angles — that would lead the robot to the goal. However, if the building is large enough, or the state-space is of a high enough dimension, such a plan would take too long to generate, and the robot would not be able to reach its goal in a reasonable time. A better strategy in this case might be to discretize the world, and plan through that grid using a search algorithm like A* [24]. The robot would then be able to take a point along the path produced by this search, and solve the optimal control problem to it as an intermediate goal. Note that depending on the dimensionality of the state space, relying on grid based discretizations may not always be the correct abstraction. If the building were large enough, or the robot had more degrees of freedom that could not be ignored, a higher level of abstraction may be necessary. This could be accomplished with a coarser level of discretization, however if the grid becomes too imprecise, the plan generated may not correspond with the true optimal path. In that case a different method of abstraction should be considered.

1.3 Navigation Through Unknown Space

We return to the problem of navigating through an unknown environment. As stated above, the planning problem involves finding a series of valid configurations that take the robot from start to goal. However, when the robot plans through a region of the environment that has not yet been explored, it is not possible to know if a configuration is valid or not (if the robot intersects with an obstacle). In order to estimate the likelihood that a plan through unknown space is valid, one possible technique would be to reason about the probability that certain unknown regions of the map are occupied by an obstacle, relying on some prior distribution of maps. Unfortunately, due to the high dimensionality of building sized maps, and the complex nature of the design of man-made structures, such a distribution over real-world environments would be exceedingly difficult to model. Due to these limitations, the best distribution that could reasonably be used would be one in which every possible map were equally likely, which is a poor representation of how man made environments are actually structured.

Even if we were to assume access to the true distribution over maps, if the planning problem is to be solved completely for the minimum cost *in expectation*, every possible environment must be considered by the robot, and each individual plan must be weighted by the likelihood of its corresponding map. It is clear that for any problem beyond trivial toy examples, it is intractable to find the optimal plan in this manner without devising some clever way to reformulate the problem. Thus, planning to reach an unseen goal requires making simplifying assumptions that reduce the dimensionality of the problem in order to plan in a higher level of abstraction.

1.4 Navigation with High-Level Actions

As discussed above, if it is not tractable to solve a planning problem at a certain level of abstraction, solving the problem at a higher level, then using that solution to create easier lower-level planning problems is a useful approach. In the context of unknown environments, using an abstraction that depends on reasoning about the state of the map in some unknown region is difficult both because of the high dimensionality of the problem, as well as the fact that the true distribution over maps is unknown. Instead, we propose planning using high-level actions, which is a different way of abstracting the planning problem.

What it means to plan using higher-level actions in the context of a navigation problem is actually quite intuitive. When solving path planning problems for ourselves, humans do not think about the low-level actions required to put one foot in front of the other. Even further, humans generally do not discretize their environment into a grid, and think about going from one grid cell to the next as they make their way down some path. Instead, when making a plan to navigate to a goal outside of direct line of sight, humans will plan at a much higher level of abstraction as depicted

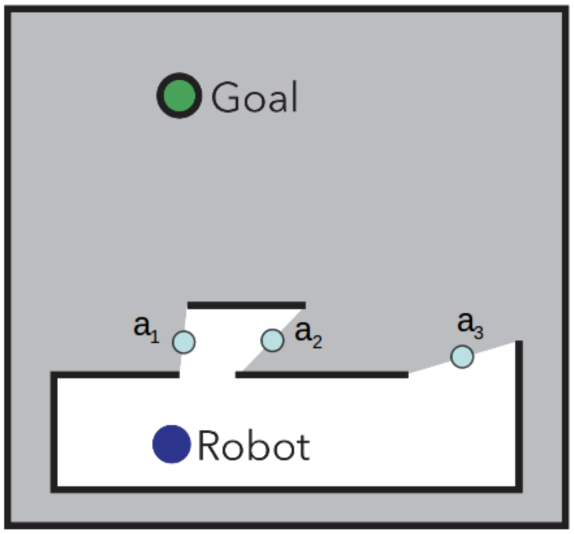

Figure 1-2: An action-centric abstraction, where each high-level action available to the robot (a_1, a_2, a_3) is represented by a boundary between free and unknown space. By reducing the action space to only these three option, it is much less computationally expensive to reason about which action to take.

in Fig 1-2. For example, if placed inside an unknown building, the actions a person might consider if they were trying to navigate through it could be to go down the hallway in either direction, or enter one of the rooms that branch off along it. It is not useful to consider each individual trajectory that goes down the hallway as a different plan because they all enter unknown space through the same "frontier," and so can be grouped together.

One benefit of using high-level actions as the basis for an abstraction is that it greatly reduces the dimensionality of the problem of planning through unknown space in a way that state-centric abstractions do not. It is generally not feasible to reason about the distribution over maps because the number of possible maps is so large. However, the possible outcomes of taking the type of action described above is that either the goal exists along that path, or it does not, meaning that if it were possible to estimate certain properties of taking an action from data, that information could be used to solve the planning problem tractably. Moreover, because of the low-dimensionality of the action space, it is significantly easier to use data-driven methods to learn such a distribution [52].

The first major contribution of this thesis is as follows. To reduce the complexity of planning in partially explored environments, a higher-level of abstraction for navigation is presented, which allows an autonomous agent to reason about plans that enter unknown space. The costs of these actions are not known exactly, and so are estimated by a neural network trained on data from environments similar to the one being navigated. Learning properties of these actions allows the abstraction to be useful for making more efficient, long-term plans.

1.5 Building a Graph of Actions

For the task of navigating to an unseen goal, the only information the robot needs in order to plan its path are the actions available to it, and the costs of taking those actions. As such, a map is sufficient for navigation if for any robot pose, it can be queried to give all available actions for that position, and their costs. Assuming the robot has the ability to generate the low-level motor commands from any high-level plan by using local sensor information, storing any further information in the map is not necessary for navigation, and so can be discarded. Thus, the minimal, sufficient map for navigation in the action-centric abstraction defined above is one that stores actions as nodes in a graph, with edges connecting actions that can be combined to form a plan through, and beyond, known space.

In order to build such a map, an autonomous agent must: 1) be able to extract high-level actions from sensor input, and 2) associate these detections from frame to frame in order to build this action-graph. To that end, the second major contribution of this thesis is a learned detector that allows the robot to extract high-level actions from vision, and a method for integrating those detections into a graph that represents what actions the robot can take at any point in the environment. Critical in this graph building process is that the agent is able to tell when an action it has detected already exists in the graph, or if it is a new action entirely. Thus, the "data-association" problem is also investigated, with two different approaches discussed in detail.

1.6 Thesis Overview

The remainder of the thesis is organized as according to the following outline:

In Chapter 2, previous approaches to solving the problem of navigating unknown environments will be discussed, in addition to other background information pertinent to this thesis. A description of the Learned Subgoal Planning approach will be presented in Chapter 3, which proposes using high-level actions to greatly simplify the problem. Chapter 4 considers the problem of extracting these high-level actions from vision, and addresses the challenge of data-association as it relates to building a sparse map for navigation. Finally, Chapter 5 summarizes the work presented in the previous four chapters, and highlights the connection between the two main contributions. Possible directions for future work are also discussed here, particularly as it relates to integrating the work presented in Chapters 3 and 4.

Chapter 2

Technical Background and Foundational Related Works

This chapter presents work in the field of mobile robotics relevant to the research that will be discussed later in this thesis. The focus of this chapter will be on previous efforts in the space of navigation in unknown environments, as well as mapping techniques, and methods to associate sparse landmark detections frame to frame. To support that discussion, background information on a variety of topics will be presented, including POMDPs (Section 2.3), different map representations for robotics (Section 2.1), and solutions to the data-association problem (Section 2.2).

2.1 Estimating and Representing the State of the World

As discussed in Chapter 1, there are two primary problems associated with navigating unknown environments. One of these is reasoning about plans which enter unknown space and how to deal with the uncertainty inherent in deciding what action to take when it is not possible to know what lies around a corner. The other problem is, once previously unknown space is revealed, how should it be represented such that a robot can plan to navigate through it.

In order to navigate through unknown environments, a robot must construct a map as it travels so as to update its belief of the state of the world. The goal of robotic mapping is to build a spacial representation of the environment from sensor measurements [62]. When navigating through a previously unexplored environment, regions of the world that were once unknown are revealed as new observations are received. In the process of searching for an unseen goal, an autonomous agent might encounter a dead-end — or otherwise decide to turn around — and be forced to double back through known space. Thus, the robot must build a representation of the world online that allows it to optimally plan trajectories through these regions in order to continue exploring. In the context of navigation, a map is useful insofar as it allows a robot to efficiently solve for the optimal plan to traverse through the space it represents.

2.1.1 Different Map Representations

Generally speaking, there are two main classes of map representations for robotics: metric maps, which encode geometric information about an environment, and topological maps, which describe the connectivity of different regions [62]. Topological representations treat "significant" regions of an environment as nodes in a graph structure, with edges indicating traversability (and often containing information about how to navigate between regions). While the sparse nature of topological maps can make planning computationally easier, they are often times much harder to build accurately in the presence of noise due to the difficulty of knowing when the robot has returned to a previously seen region [62]. Moreover, the trajectories produced by planning through such maps may not make it possible to find optimal low-level paths between the elements of the higher-level plan [61]. As such, the majority of maps utilized by robots in practice are metric, or some combination of both known as "topometric." The following discussion presents several examples of metric and topometric maps that have been developed over the last few decades of robotics.

One of the most common map representations used in robotics (and one of the first hypothesized) is the occupancy grid. Developed by Elfes and Moravec in the

1980's, occupancy grid mapping is a popular way to convert depth measurements into a dense representation of obstacles in the world [15, 16, 44]. Occupancy grids probabilistically represent the world as a discretized grid, where each cell is either free space, occupied by an obstacle, or unobserved. An advantage of this representation is that it is well understood how to plan through such a grid [12, 24]. Furthermore, because it is capable of representing unexplored space, it is possible to generate plans through an occupancy grid which can be useful for navigation through partially explored environments. In fact, the work in Chapter 3 takes advantage of this property. However, though there exist systems to estimate depth from monocular camera images, without specialized hardware or stereo cameras, it can be difficult to reliably construct an occupancy grid on a robot. Moreover, as the size of the map grows, it can be expensive to maintain such a dense representation of the environment, so a more minimal representation would be preferable.

The type of map best suited for a particular robot depends in part on the way it perceives the world — i.e., the sensors available to it — and the task it is trying to accomplish. Consider a robot equipped with a sensor that is able to determine the distance of obstacles within direct line of sight. Hardware like LIDAR or SONAR are examples of such specialized hardware that provide this information and are common in the field of robotics [36]. Similarly, range information can be extracted from depth cameras (RGB-D), or even a well-tuned stereo camera system. Systems such as these are well suited to build an occupancy grid representation of the world. However, if a robot is equipped with a monocular camera (a lightweight, inexpensive alternative to laser range sensors), or if it is limited computationally, representations other than occupancy grids are generally more popular.

An alternative to dense representations such as the occupancy grid are sparse maps, which detect landmarks in space and track them from frame to frame. Landmarks can be defined as any point in physical space the robot is able to detect and keep track of in its map. What these landmarks are, as discussed above, depend on the sensors available to the robot, but can be utilized in both topometric [50], and metric approaches [43].

Consider the maps generated by modern visual SLAM (Simultaneous Localization and Mapping) algorithms. The goal of these systems is to estimate a robot's trajectory through space from camera input, and so build a factor graph of landmarks and poses which is optimized using bundle adjustment [65]. So called "sparse" methods such as ORB-SLAM detect visual features — in this case ORB features [55] — and maintain a point-cloud of these detections as the landmarks [45]. A benefit of these systems is that, because there are so many landmarks maintained in the map, localization within that map can be highly accurate. However, there are several drawbacks. One particularly relevant to the work presented in this thesis is that there is no notion of unknown space in these representations because the map is just a collection of points in space. Thus, while extremely useful for localization, these maps are not easily used for building plans that reason about paths through unexplored parts of the environment. Additionally, because there may be regions where there are no visible features, collision checking in sparse maps is difficult.

One map representation which does specifically encode the actions a robot might take to explore its environment is Gap Navigation Trees [64], which construct a minimal map representation of gap detections — discontinuities in depth from the perspective of the robot — and traversed space. The resulting tree-structured map is designed with navigation in mind: the map's *unexplored nodes* correspond to detected gaps the robot has never revealed, and each of these special nodes corresponds to a route through which the robot can enter unknown space in an effort to find the unseen goal. However, Gap Navigation Trees are notably limited to navigation in simply-connected environments [46] and require an infinitely-precise, noiseless sensor, thus limiting their utility in practice. Chapter 4 presents a novel map representation that keeps track of unknown space by maintaining a graph of high-level actions with monocular visual input in order to enable high-level planning. As with all maps however, being robust to noise in the system is something that must be considered.

2.1.2 Probabilistic Mapping

There are two primary challenges facing a robot that is trying to build a map as it travels, both of which are directly related to noise in the system. Noise in state-estimation means that the robot may not have an accurate estimate of its position in the world relative to a global reference frame. This uncertainty is compounded by the fact that there also may exist noise in sensor measurements, meaning each observation does not necessarily represent the true state of the world. In order to deal with this reality, the robotics community has converged on probabilistic mapping techniques that explicitly reason about uncertainty [62].

Over the past few decades, the use of probabilistic methods in mapping in the field of robotics has greatly matured. Initially, filtering techniques like the Extended Kalman Filter (EKF) were relied upon to update the estimated location of landmarks (physical points in 3D space) in a map [56, 7, 21, 17]. While effective in some cases, early forms of this approach only maintain the most likely estimate of the map, updated at each time step. Thus, they are unable to recover if some error was made in the map building process. Rao-Blackwellized particle filters are another popular approach, which represent states of the world as particles with different weights, and so are better able to represent multiple hypotheses of the world in cases where observations are uncertain [13, 43]. However, issues like particle death make it difficult to correct for errors in the case of drift over large time scales. Additionally, storing and constantly updating multiple different maps can be overly computationally expensive as the size of maps grow.

It is important to note that these approaches were developed primarily in an effort to solve the SLAM problem (Simultaneous Localization and Mapping). SLAM entails both estimating properties of the map, as well as the robot's position in that map. However, by their very nature, filtering approaches are imperfect in solving SLAM because they do not update estimates of past poses with new measurement information, which makes the problem of loop closure more difficult [62]. The reason this was not done in the past was, for the most part, due to limits in available computation. However, recent state-of-the-art SLAM systems have begun to rely on smoothing approaches to incorporate all measurements into estimates. One algorithm that has been extensively used in modern systems is bundle-adjustment [65], which is a method for solving the non-linear least squares problem of optimizing the 3D estimate of each pose and measured landmark simultaneously. Compared to filtering approaches, which marginalize out past pose and sensor information, solving the full optimization produces a better solution to the SLAM problem [57].

In Chapter 4, a novel map representation is presented that deals with noise by maintaining a posterior over the space of possible maps based on sensor observations. Then, by comparing the likelihood of different map proposals using this posterior, it can determine the most likely map. This technique is possible due to how sparse the features in the map are, and would not be feasible for maps composed of point-clouds such as those is modern visual SLAM systems.

2.2 Data-Association in Mapping

The second major challenge in robotic mapping is also a direct result of measurement noise, specifically as it relates to the so called data-association problem (also known as the correspondence problem). The data-association problem refers to the challenge of determining if a measurement at one instance in time corresponds to the same physical entity as a measurement at some other time. For example, if a robot is equipped with a sensor that detects doors, and it sees a door in two different images, the data-association problem refers to the task of determining if those two detections are of the same door. If there were no noise in the system, data-association could be solved easily by simple checking where in space each detection is, and comparing it for exact matches, though this is not realistic.

How the data-association problem is solved depends on the type of map representation a robot builds. For example, because an occupancy grid is a dense representation of the world, data-association is ignored when the robot localizes itself in the map. This is because each measurement informs the value of a certain number of grid cells, which depends only on the robot's position within the map and the measured ranges. For sparse maps however, each time a landmark is detected, the robot must determine if that landmark already exists in the map it is building, or if it is a new landmark that must be added.

The goal of data-association is to determine if a detection is one the robot has already seen before. One intuitive approach is to define a distance metric from each new landmark detection to landmarks that exist in the map. From there, it is a simple matter of determining the landmark that is closest to the new detection according to this distance metric, and, if it is within a certain threshold, update the estimated position of that landmark with the new observation [48, 61]. This leaves three questions to be answered in regards to data association. 1) How do we choose the distance metric? 2) How do we update the map assuming we can have multiple detections in each observation? 3) How can we be robust to errors in data association?

2.2.1 Distance Metrics

How to choose the distance metric depends on the information available to a robot. For example, if all that is known about a landmark is its position in space, Euclidean distance may be the only option. This is often the case with the types of visual features utilized in many sparse visual SLAM systems [45]. For each feature detected in a particular frame, a so-called "descriptor" – a fixed-dimensional embedding meant to distinguish this feature from others — is computed. How these descriptors are computed depends on the type of feature detected [37, 5, 55], but in general, they are compared to descriptors of past features, and if they are close enough in Euclidean distance according to some threshold, the features are matched [45]. In this framework Euclidean distance can be an effective metric, as long as the dimensionality of the descriptor is complex enough.

Recent developments in the field of learned descriptors have also shown promise. Learned descriptors, like their hand-crafted counterparts, are mappings from an image-patch to a fixed-dimensional space. Unlike hand-crafted descriptors however, a learned feature descriptor uses a convolutional neural network for its mapping. Recent works, like “L2-Net” [63] (published in 2017), show improvement over well established traditional descriptors for certain data-sets. If the descriptors effectively differentiate between unique points in space and are invariant to the orientation and distance of the camera from the feature, feature matching can be an effective way to solve the data-association problem.

However, in maps where visual features are not the landmarks, it is not as well understood how to compute descriptors. Instead of comparing distances in some high dimensional feature space, systems that detect physical landmarks often estimate the 3D pose of objects, and measure distance in that space. With a different observation model, metrics that incorporate noise can be used as well. For example, one popular approach is to model each landmark as a Gaussian, and to use the Mahalanobis distance [38] as the distance metric [46]. In principle however, any metric can be used as long as the information stored in the landmark and observation allow for it to be computed. In Chapter 4, both a Mahalanobis distance based approach, and a learned association metric based on learned feature descriptors are compared in the context of building a map from vision to be used for navigation.

2.2.2 Data-Association with Multiple Detections

Regardless of the metric, given all information, for single hypothesis data-association, the optimal technique would be to find the maximum likelihood association for each detection. In other words, associate each detection with the landmark that was most likely to generate it [42]. In the case where there are multiple detections at a given time, there may be instances where two different detections are most likely to be associated with the same landmark, and are within the threshold for association. This can happen when two landmarks are particularly close in terms of the distance metric, even though they are distinct. In this case, the concept of mutual exclusion [42] — that each landmark can only be associated with one detection per observation — is useful for overcoming an imperfect distance metric.

In order to ensure that the optimal associations are found, even in the case where multiple detections meet the criteria for association with the same landmark, certain detections that would otherwise be associated with a landmark will have to search for a different correspondence (or form a new landmark). The Hungarian algorithm is known to be able to solve for the optimal assignments for all detections in polynomial time [33]. The Hungarian algorithm is a combinatorial optimization algorithm, which can be modified to find the optimal solution in $O(n^3)$ [33]. If the number of landmarks in the map are of too great a number however, some implementations use greedy approaches in order to reduce computational load [42]. This is true particularly in maps built from visual features, and can lead to errors in data-association.

2.2.3 Errors in Data-Association

For any system in the presence of noise, there will likely be cases where landmarks are associated incorrectly. The cause of these errors can be from noise in the system (either in observations or localization), a distance metric that is an imperfect model, false-positive/negative detections of landmarks, or shortcuts taken to reduce computational load. The problems caused by incorrect data-association can be very difficult to overcome. If detections are incorrectly added to the map, or landmarks are updated with detections that do not correspond to them in reality, the map the robot builds will not accurately represent the underlying environment. These errors can compound, and lead to a robot failing to reach its goal. For a robot tasked with navigating to an unseen goal, this can mean that certain routes might be blocked off to it, preventing it from accomplishing the task.

How errors in data-association are dealt with once again depends upon the type of map a robot is building. Maps composed of 3D point-clouds, due to the fact that they track so many landmarks, rely on approaches such as RANSAC (random sample consensus) to prune failures in association [45]. Proposed by Fischler and Bolles in 1981, RANSAC [18] is a method to remove outliers from data by iterative removing points at random to see if the remaining data better fits some model. Though incredibly popular as a method for correcting data-association in visual SLAM, it is stochastic, and can be computationally expensive for large numbers of features. Moreover, once an outlier is removed via RANSAC, it can not be added back into the graph, meaning if something is removed incorrectly, the map will not be able to accurately represent the environment.

Other methods for overcoming errors in data-association include maintaining multiple hypotheses of the world, weighted by their probability. In this way, the robot can plan using the most likely map, while still being able to switch to a map that made a different choice in data association if new measurements dictate. This is the approach taken in particle filter based SLAM system [43, 50], as well as more modern, factor graph based approaches [19]. An unfortunate downside is that these methods can be very computationally expensive, and so may be difficult to deploy on a robot operating in real-time.

Data-association is a critical module in any sparse map building procedure. In Chapter 4, a map representation is presented that can be built from monocular vision, and which allows a robot to plan via high-level actions. This map is shown to be robust to noise in both the learned action detector as well as in data-association. Additionally, different methods for data-association are considered in the context of building a graph of high-level actions that a robot can take in order to navigate through unknown space.

2.3 Navigating Under Uncertainty

For a robot to be able to interact with the world, it must have a model of both itself and its environment. The best choice of this model is dependant on the task the robot is trying to accomplish, as well as what information about the world is available to it. If a robot has full knowledge of its environment, it would be able to generate a perfect map in order to represent the world, with no regions that are unexplored. When tasked with navigating through an unknown world however, an autonomous agent must construct its map as it travels.

With an incomplete map, it is impossible for a robot to decide how to navigate optimally with complete confidence. How could a robot be expected to find the best trajectory to a goal when it can not be sure what lies around each corner? In order to solve this problem, an autonomous agent must be able to incorporate this uncertainty into its model of the world.

In an ideal world, the navigation problem could be modeled as a sequential decision making process with perfect information. First the robot receives an observation of its surroundings, then decides how best to act. After it has taken that action, the process repeats until the robot reaches its goal. If an observation contains information about the full state of the world, and that world is deterministic, then the robot can plan to navigate optimally. If we make the assumption that the optimal plan from a given state is independent of previous states — the Markov assumption [39] — this problem is well modeled by a MDP (Markov Decision Process), and can be solved using such techniques as value or policy iteration [6].

Unfortunately, the types of sensors available to a mobile robot are limited in a way that prevents the problem from being framed simply as an MDP: they can only observe what is in direct line of sight and within some limited range [25]. Laser range sensors are only reliable up to a certain distance, and neither they nor cameras can "see" through walls. Thus, even if a robot's sensors are perfectly noise free, and detect everything within their range exactly, there will inevitably be regions of the environment where the robot has incomplete information. In order to account for this, we must incorporate the fact that each observation only includes partial information about the environment in our model.

2.3.1 Navigation as a POMDP

The problem of autonomously navigating unknown environments can be formulated as a Partially Observable Markov Decision Process (POMDP) — a generalization of an MDP [28]. In an MDP, an agent knows exactly the state of the world, what actions it can take for each state, and the possible outcomes of taking a given action. In a POMDP however, the agent is not certain of the state of the world, and instead has some belief, represented as a distribution over all possible states, that is informed by observations [28]. If we define the state of the world as a static map plus the robot's position within that map, we can represent the problem of navigating unknown environments in this framework by defining the belief of unexplored regions as uncertain.

The following tuple can be used to represent the different elements of a POMDP: (S, A, T, R, O, Ω), each of which is described below in the context of navigation of unknown environments.

1. S State: The state represents both the internal state of the robot, and the state of the world, usually represented by a map. Depending on the robot, a robot's state can have many components such as various joint angles and velocities, or be as simple as a pose in 2D. The state of the map, which can be continuous or discretized (like an occupancy grid), is unchanging. Note that, in this work, we are only concerned with discrete state-spaces. What distinguishes a POMDP from a standard MDP however, is that in the case of an MDP, the state is known perfectly, while this is not the case in a POMDP. For example, for the problem of the navigation of unknown environments, even with a perfect sensor, due to limitations of range and occlusion, not all of the map's state is known at all times, because not all of the map has been seen by the robot's sensors.

2. A Actions: The actions available to the robot at any given instance. An action takes the robot from one state to another with some probability. In many robotics problems, the action set is some low-level control, like changing the steering angle or velocity. In this work however, we present a method for simplifying the problem by considering a set of high-level actions. This will be discussed in greater detail in chapter 3. In general, by solving a POMDP, the robot will know for any given state what the optimal action is with respect to some objective function.

3. T Transition: The probability that taking a certain action will result in the robot being in a given state. For example, if a wheeled robot is directed to make a turn, there may be some probability that the wheels will slip, and the robot will end up in a different state than what was intended.

4. R Reward: The reward (sometimes formulated as a cost) is an element in the agent's objective function that the autonomous agent is maximizing (or minimizing

in the case of cost) to solve the POMDP. Rewards can be given for taking certain actions, or for reaching intermediate states, though in the context of this work, the navigation task is formulated as a min-distance problem. In other words, the optimal path a robot can take is the one that reaches the goal in the shortest possible distance.

5. Ω Observation: An observation is all information that an agent receives about the world at a given time. The observation contains information about the state of the world, and in the case of a POMDP, is accompanied by some uncertainty. For the case of a discretized state-space, an observation contains information about which cells in the grid are occupied by obstacles.

6. O Conditional Observation Probability: The conditional observation probability represents the probability that a given observation correctly represents the world. The inaccuracy in an observation can come from a noisy sensor, though in this work we are primarily concerned with incomplete observations received by a mobile robot. For example, if the robot receives an observation from a laser-range sensor, it only has information about the environment that is in direct line of sight from the sensor. Obstacles occlude whatever is behind them, so information in that region which is occluded would have a conditional observation probability of zero.

2.3.2 Belief States and the Bellman Equation

The defining characteristic of a POMDP is that the agent does not have full knowledge of the state of the world at all times, and so must maintain a belief of the true state based on its observations. The belief at a particular state $b[s_t]$ can be defined as the pr obability that s represents the true state at a given time. It has been shown that the belief is a sufficient statistic for for all past observations and the initial prior belief state of an agent, meaning the distribution encodes all information needed to act under uncertainty [28, 3].

As mentioned above, navigation through unknown environments is often modeled as a POMDP, where the belief state of unobserved regions is uncertain. To "solve" such a problem, the agent searches for the best action it can take in a given state, with the goal of optimizing some objective function. The expected cost of an action under the optimal policy can be computed recursively using the Bellman Equation [6]:

$$Q^*(b_t, a_t \in \mathcal{A}(b_t)) = \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t) \left[R(b_{t+1}, b_t, a_t) + \min_{a_{t+1} \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a_{t+1}) \right] \tag{2.1}$$

where $R(b_{t+1}, b_t, a_t)$ is the expected cost of reaching belief state b_{t+1} from b_t by taking action a_t. We can define the belief state to be a two-element tuple consisting of the partially observed map m_t and the robot pose q_t: $b_t = \{m_t, q_t\}$.

In order to highlight the focus of the research in chapter 3, we make some simplifying assumptions. First among these is that we have a noiseless sensor, meaning that everything that is within sight of our robot can be perfectly detected. Additionally, we assume that, at the lowest level of abstraction, we have a deterministic transition function. In other words, this means that every action the robot takes will be executed perfectly (e.g., no wheel slip), though this may not necessarily be true at higher levels of abstraction. Finally, we do not attempt to solve the problem of localization in this work, and assume the position of the robot relative to some global reference frame is known perfectly.

Even with these simplifying assumptions however, the problem is still extremely difficult to solve. The robot can only observe what is within its direct line of sight, and so cannot update its belief of the entire map with each observation. If the goal exists outside of the region the agent has seen, it must in some way attempt to solve the POMDP in order to plan to that goal even with this incomplete information. Unfortunately, in practice, computing the expected cost via Eq. (2.1) is intractable, both because the complexity scales poorly for large problems, and because it requires taking an expectation over the distribution of all possible maps.

2.4 Solving the POMDP

Work on solving navigation and exploration tasks in uncertain environments using POMDPs dates to the mid-1990's [28, 35, 60]. Due to the computational challenges of solving POMDPs however, most techniques established through 1990 were too inefficient to be used on anything beyond very simple cases (2-5 states at most) [8]. In 1994, Michael Littman pushed the field forward when he published his work on the "Witness Algorithm," which is capable of solving POMDPs in environments with up to 16 states exactly [34].

Later, work from Littman, Cassandra, and Kaelbling compared methods for solving POMDPs approximately using such techniques as truncating the evaluation of the witness algorithm, solving the underlying MDP by ignoring the observation model (Q_{MDP}), and applying different Q-Learning techniques from reinforcement learning literature [35]. They found these methods were more or less comparable, though thoroughly unable to solve problems of as few as even 57 states. Through combining techniques however (specifically by seeding Q-learning with Q_{MDP}), the authors were able to satisfactorily approximate the solutions to POMDPs with "nearly 100 states" [35].

Despite these advancements however, as the size of environments expands beyond simple toy examples, these methods are no longer tractable for a robot operating in real time. Instead of solving the POMDP completely, recent efforts have been made to approximate it to solve problems of a larger scale.

2.4.1 Approximating short-horizon POMDPs for Navigation

Recent efforts in approximately solving POMDPs online involve solving smaller problems which yield a policy similar to the solution to the full POMDP. DESPOT searches a set of randomly sampled *scenarios*, which helps alleviate the problems associated with solving POMDPs in high-dimensional state-spaces [67]. The algorithm finds an approximately optimal policy by simulating the execution of all policies in the belief tree of sparcified samples. The authors show they can find an estimate of the true value of a policy even in large state-spaces given that a good small policy exists. However, long-term planning in the context of navigation in this framework is still difficult.

While the goal of this thesis is the navigation of large-scale environments, work in navigating short-horizon navigation problems can be insightful. Recently, progress has been made in searching for collision-free paths through partially explored environments, wherein an autonomous agent must reason about unknown space, albeit on a smaller scale than the work presented in this thesis. Karaman and Frazzoli [31] navigate unknown random forest environments by making strong assumptions about the environment distribution. Since they know the locations of trees are generated by a homogeneous Poisson process, and know the dynamics of their agent (modeled after a bird at high-speeds), the authors are able to bound collision probabilities for given speeds. Some recent work has instead focused on using a dynamic action set [54] or boundaries between free and unknown space [2, 48] for navigation/exploration of unknown environments.

If the distribution over states is not known, learning is required to estimate the expected cost [52]. Most literature that uses learning to explore unknown environments focuses on short time horizon planning. Richter *et al.* [51] learn to solve an approximate POMDP for navigation of unknown environments, yet restrict their planning horizon to only a few time-steps. In Richter and Roy's later work [53], the authors leverage supervised learning to estimate the collision probabilities of trajectories that enter unknown regions of the map in order to navigate structured environments more quickly. Though predictions are limited to short-term trajectories, these publications were inspirational in a number of ways. First, the authors utilize a discrete action set of 50 different actions as an action-centric abstraction for planning. Second, approximations are made to the Bellman equation to allow it to be factored into components (such as collision probability) that can be estimated via learning.

2.4.2 Navigation using Deep Reinforcement Learning

While progress has been made in utilizing the Bellman equation to plan through unknown space, the deep reinforcement learning (Deep RL) community has been making rapid progress towards model-free navigation of unknown environments [22, 58, 68, 41, 14]. Once again however, most such research focuses on small environments or, as in Kahn *et al.* [29], use time-horizons insufficient for making reasoned decisions in environments of our size, which can require recall of information over thousands of steps.

The MERLIN agent [66] uses a differentiable neural computer to tackle maze navigation and other goal-oriented tasks, which enables storage and recall of information over much longer time-horizons than is typically realizable by standard "end-to-end" Deep RL systems. However, their approach requires orders of magnitude more data than is used in the approach described in Chapter 3 (billions of samples versus hundreds-of-thousands) and on environments considerably smaller than ours, owing to the difficulties associated with model-free reinforcement learning with delayed rewards [11, 32]. Like other Deep RL agents, they also lack guarantees their agent will ultimately succeed at the assigned task.

The work presented in Chapter 3 expands upon the work that utilizes supervised learning to plan into unknown space, with a focus on long-horizon planning. The primary contributions are an abstraction that allows for efficient planning, and a factorization of the Bellman equation that enables reasoning about long term plans. Chapter 4 will present an alternative to the mapping techniques discussed earlier in order to enable efficient navigation of unknown environments with visual input.

Chapter 3

Learned Subgoal Planner

As stated Chapter 1, in order to navigate to an unseen goal, an autonomous agent must reason about plans through unknown space. Returning to the motivating example from Chapter 1, we imagine a robot tasked with navigating a previously unexplored building, adding to its partial map of the environment as it travels. Reaching its goal requires the robot to generate plans which enter unknown parts of the map, and in so doing, reason about the cost of trajectories through space which has not yet been observed. As was discussed in Chapter 2, past solutions to problems like this one avoid the difficulties associated with reasoning about unknown space by optimistically assuming all unobserved regions of the map are free of obstacles [4, 20]. However, attempting to travel across a building using this approach will lead such an optimistic planner to make sub-optimal decisions while navigating. If tasked with reaching a far away conference room, for example, we would expect a naive robot to encounter many dead-ends and other local minima, greatly increasing travel time. If we aim to navigate partially-revealed environments in minimum distance, the robot will need to reason more intelligently about unknown space.

3.1 Planning Using High-Level Actions

Planning more intelligently requires making inferences about environmental topology, so that likely dead-ends may be predicted and avoided. For example, offices in a building usually only have a single entrance and very rarely connect different regions, so any proposed motion plans that attempt to reach a faraway goal via an unexplored office should have a higher expected cost than those that plan to navigate through hallways. We have shown that navigating in a partially-revealed environment can be modeled as a Partially Observable Markov Decision Process [28] (POMDP), which can be leveraged to compute the expected minimum cost path. However, using this model to evaluate the expected costs of actions, like comparing trajectories which go in different directions down a hallway, is impossible in practice. Not only does this comparison require enormous computational effort for large maps, but it also necessitates access to a distribution over possible environments. Obtaining such a distribution in a form that is useful for navigating unknown space is difficult in practice for complex environments.

Despite the advances of learning-based approaches for navigating in unknown environments [51, 22, 68], there are still limitations. One key drawback of many approaches so far has been in their ability to learn policies that reason over long time horizons, since previous efforts have focused predominantly on short-horizon costs. Methods such as those presented in Richter *et al.* [51, 52] and Kahn *et al.* [29], while useful for avoiding obstacles, are insufficient for recognizing and avoiding the long-term implications of actions, which may impact navigation over hundreds of meters. Those approaches which use reinforcement learning to plan over longer time horizons suffer from practical limitations associated with data complexity [11, 66] or delayed rewards [32], and so make it difficult for such agents to learn policies that can effectively navigate large-scale, unknown environments.

In order to reason about the structure of an unobserved environment beyond the short term more easily, we introduce an abstraction to the planning problem that enables us to efficiently make predictions about actions which lead the robot to enter unknown space. To that end, we associate "subgoals" in the planning problem with each boundary between free and unknown space in our map. We define a dynamic action set where each high-level action corresponds to the robot traveling to the goal through a particular subgoal. Using subgoals to define the set of actions, we begin with the Bellman equation to derive an algorithm that allows us to use those actions to plan. By reducing the complexity of the action set in this way, we are able to avoid the computational challenges of reasoning over individual trajectories or motion-primitives, and estimate the expected cost of navigating to the goal via one of the subgoals.

Representing the possible paths to the goal, one of the subgoals must be passed through for the robot to enter unknown space and reach the unseen goal. Thus, the set of subgoals represents all possible avenues for exploration of the environment. The set of dynamic actions (subgoals) is computed on-line from the map and updates as the robot explores. Using a neural network, our approach learns the properties of the unknown environment beyond each subgoal: e.g., the likelihood that attempting to reach the goal via a subgoal will ultimately lead to the goal or to a dead end. By learning these values, our algorithm is able to incorporate prior information from similarly structured environments, and estimate the long-time-horizon cost of our actions, while avoiding the computational costs usually incurred when planning under uncertainty.

Our approach enables experience-guided navigation of unknown environments in real-time on an autonomous platform. Moreover, once a high-level action is chosen, we use classical planning techniques to navigate toward each subgoal, meaning our planner is guaranteed to reach the goal if a feasible path exists, even in unfamiliar environments where learning may not provide accurate predictions. We demonstrate the effectiveness of our technique in simulation by showing that the expected cost of using our planner for navigation is 21% lower on real-world floor plans as compared to a standard optimistic heuristic. We also include real-world experiments on an RC car and show promising performance consistent with the simulated results (Section 3.4), thus demonstrating that our algorithm is suitable for operation on a real-world robot.

3.2 Planning with Subgoals

When navigating to a goal in a static, unknown environment, a robot's objective is to minimize the distance it will travel in expectation. As stated in Chapter 2, navigating an unknown environment can be modeled as a Partially Observable Markov Decision Process. Using this framework, the expected cost of an action under the optimal policy can be computed recursively using the Bellman Equation, re-presented here from 2.1:

$$Q^*(b_t, a_t \in \mathcal{A}(b_t)) = \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t) \left[R(b_{t+1}, b_t, a_t) + \min_{a_{t+1} \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a_{t+1}) \right] \tag{3.1}$$

As before, $R(b_{t+1}, b_t, a_t)$ is the expected cost of reaching belief state b_{t+1} from b_t by taking action a_t. For a robot navigating while building a map, the belief state can be represented as a two-element tuple consisting of the partially observed map m_t and the robot pose q_t: $b_t = \{m_t, q_t\}$.

To simplify the calculation of the expected cost, we introduce our Learned Subgoal Planning algorithm, which we use to evaluate the expected cost of selecting a motion plan that passes through a particular subgoal. In our abstraction, each subgoal corresponds to a boundary between free and unknown space, and a plan through a particular subgoal represents a path towards the goal which enters unknown space through that boundary.

At the time of planning, the subgoals define the set of actions available to a robot, where each action corresponds to traveling to the goal via the unknown space that exists beyond the frontier of a particular subgoal. The action loop for our subgoal planning agent is as follows: (1) we obtain the set of subgoals from the map (this yields a set of actions $\mathcal{A}(b_t)$), (2) we compute the expected cost of selecting an action a_t using the process described in Section 3.2.1, (3) we choose the subgoal q_g^* that minimizes the expected cost of navigating to the goal, (4) we compute a motion plan passing through subgoal q_g^*, and (5) we select a low-level motion primitive that moves along the computed path. This repeats at each time step until the goal is within the

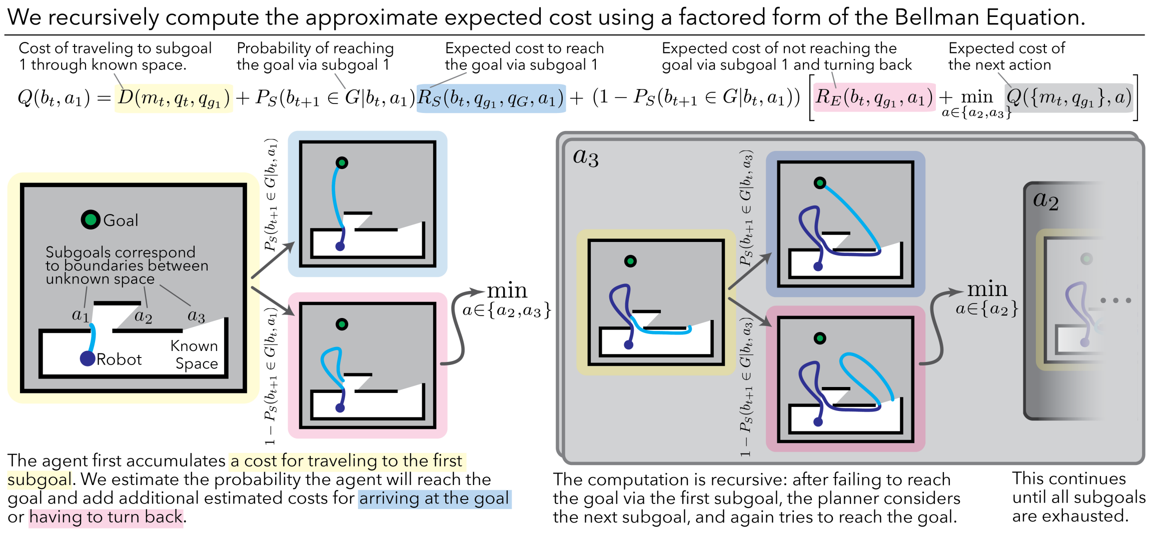
We recursively compute the approximate expected cost using a factored form of the Bellman Equation.
Cost of traveling to subgoal 1 through known space.
Probability of reaching the goal via subgoal 1
Expected cost to reach the goal via subgoal 1
Expected cost of not reaching the goal via subgoal 1 and turning back
Expected cost of the next action
$Q(b_t, a_1) = D(m_t, q_t, q_{g_1}) + P_S(b_{t+1} \in G|b_t, a_1) R_S(b_t, q_{g_1}, q_G, a_1) + (1 - P_S(b_{t+1} \in G|b_t, a_1)) \left[R_E(b_t, q_{g_1}, a_1) + \min_{a \in \{a_2, a_3\}} Q(\{m_t, q_{g_1}\}, a) \right]$
Goal
Subgoals correspond to boundaries between unknown space
a_1
a_2
a_3
Robot
Known Space
$P_S(b_{t+1} \in G|b_t, a_1)$
$1 - P_S(b_{t+1} \in G|b_t, a_1)$
$\min_{a \in \{a_2, a_3\}}$
a_3
$P_S(b_{t+1} \in G|b_t, a_3)$
$1 - P_S(b_{t+1} \in G|b_t, a_3)$
$\min_{a \in \{a_2\}}$
a_2
...
The agent first accumulates a cost for traveling to the first subgoal. We estimate the probability the agent will reach the goal and add additional estimated costs for arriving at the goal or having to turn back.
The computation is recursive: after failing to reach the goal via the first subgoal, the planner considers the next subgoal, and again tries to reach the goal.
This continues until all subgoals are exhausted.

Figure 3-1: This schematic gives an overview of our subgoal planning algorithm, which allows us to plan through an unknown environment by computing the expected cost of each action in a way that is computationally feasible. Learning is used to estimate the terms P_S, R_S and R_E, thereby introducing prior knowledge about the environment class of interest into the decision-making procedure. This equation is derived and discussed further in Sec. 3.2.1.

revealed map, and the agent is able to plan to it directly.

3.2.1 Approximating Expected Cost via the Subgoal Planner

In order to take advantage of our action-centric abstraction, we first factor the Bellman Equation according to our abstraction and introduce terms we later estimate with learning. Without loss of generality, we can split the future belief states b_{t+1} into two sets: future states in which the robot has reached the goal, which we write as $b_{t+1} \in G$, and future states in which it has not, $b_{t+1} \notin G$. This reduction of the state space allows for the problem to be computationally tractable, and is possible because of the high-level action set. Noting that the future cost is identically zero for states that reach the goal (no more actions are necessary once the robot is in the goal state), we can eliminate $Q(b_{t+1}, a_{t+1})$ when $b_{t+1} \in G$ and rewrite Eq. (3.1) as follows:

$$\begin{split} Q(b_t, a_t \in \mathcal{A}(b_t)) = P_S(b_{t+1} \in G|b_t, a_t) \sum_{b_{t+1} \in G} P_G(b_{t+1}|b_t, a_t) R(b_{t+1}, b_t, a_t) + \\ (1 - P_S(b_{t+1} \in G|b_t, a_t)) \sum_{b_{t+1} \notin G} P_{\bar{G}}(b_{t+1}|b_t, a_t) \left[R(b_{t+1}, b_t, a_t) + \min_{a \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a) \right], \end{split} \tag{3.2}$$

where $P_S(b_{t+1} \in G|b_t, a_t) \equiv \sum_{b_{t+1} \in G} P(b_{t+1}|b_t, a_t)$ is the proportion of states in which the agent successfully reaches the goal after selecting action a_t from belief state b_t and $P_G(b_{t+1}|b_t, a_t) \equiv P(b_{t+1}|b_t, a_t)/P_S(b_{t+1} \in G|b_t, a_t)$ is normalized according to $\sum_{b_{t+1} \in G} P_G(b_{t+1}|b_t, a_t) = 1$, so that the first term in Eq. (3.2) can be thought of as the expected cost over states that reach the goal times the proportion of states that reach the goal (both of which we estimate using learning - Section 3.2.2). $P_{\bar{G}}$ is defined in the similar way, though for states which do not successfully reach the goal. Evaluating the recursive term in Eq. (3.2) remains particularly difficult to compute, because it implies knowledge about the actions which will be available to the robot after the selected action has been taken. This requires predicting the influence of taking an action on other actions. We reduce this term with a simplifying assumption: that the expected cost of the future states can be approximated using the current observed map m_t. Specifically, we assume that the set of actions which remain after one has been selected is simply the current set of actions minus the action that was just chosen: $\mathcal{A}(b_{t+1}) = \mathcal{A}(b_t) \setminus \{a_t\}$. These assumptions make computing the expected cost of an action significantly easier, since the recursive term in the resulting approximate Bellman Equation only depends on the current observed map m_t:

$$\min_{a \in \mathcal{A}(b_{t+1})} Q(b_{t+1}, a) \approx \min_{a \in \mathcal{A}(b_t) \setminus a_t} Q(\{m_t, q_{t+1}\}, a), \tag{3.3}$$

where $\{m_t, q_{t+1}\}$ is a belief state consisting of the *current* observed map state m_t and the *next* robot pose q_{t+1}. To compute the expected cost-to-go at time t via our subgoal based abstraction, we first compute the set of actions, $\mathcal{A}(b_t)$, which each correspond to planning to the goal through one of the available subgoals. Following Eqs. (3.2) & (3.3), the expected cost can be written as,

$$\begin{aligned} Q(b_t, a_t) = &\overbrace{D(m_t, q_t, q_g)}^{1.} + P_S(b_{t+1} \in G|b_t, a_t) \overbrace{R_S(b_t, q_g, q_G, a_t)}^{2.} + \\ &(1 - P_S(b_{t+1} \in G|b_t, a_t)) \left[\underbrace{R_E(b_t, q_g, a_t)}_{3.} + \min_{a \in \mathcal{A}(b_t) \setminus a_t} \underbrace{Q(\{m_t, q_g\}, a)}_{4.} \right], \end{aligned} \tag{3.4}$$

where we define the following terms:

1. $D(m_t, q_t, q_g)$: the cost of traveling from q_t, the current location of the agent, to a subgoal q_g (as specified by an action a_t). Determining this cost is as simple as planning the shortest cost path through the known map from the robot to the chosen subgoal. Since this motion plan occurs entirely within known space, we use Dijkstra's algorithm as a heuristic to calculate this cost. Note that in the recursive part of this cost function, $D(m_t, q_t, q_g)$ becomes the cost of traveling from one subgoal to the next. This represents the cost of trying the next action after one has failed to find the goal.

2. $R_S(b_t, q_g, q_G, a_t)$: the expected cost of success. Suppose the robot succeeds in reaching the goal via the subgoal specified by action a_t with a probability $P_S(b_{t+1} \in G|b_t, a_t)$. Then, with probability P_S, we accrue an expected cost of traveling from the subgoal q_g to the goal q_G via unknown space.

3. $R_E(b_t, q_g, a_t)$: the exploration cost. In the event that the agent does not reach the goal, we say that it instead *explores* the space beyond the subgoal of interest. With probability $1 - P_S$, the agent then accumulates an *exploration cost*, the expected cost of trying to navigate to the goal via a_t and returning to the subgoal q_g.

4. $Q(\{m_t, q_g\}, a)$: the future expected cost. If the robot does fail to reach the goal through a particular subgoal q_g, this represents the cost of considering a different action. After exhausting one action, we choose a new action from the reduced set, $a \in \mathcal{A}(b_t) \setminus a_t$. We compute this final term recursively until the costs of all actions in $\mathcal{A}(b_t)$ have been computed.

An illustration of our procedure for a simple example can be found in Fig. 3-1.

3.2.2 Learning Properties of Unknown Space

The proposed abstraction has greatly reduced the complexity of the planning problem. Given access to the terms in Eq. (3.4) — the probability of succeeding in reaching the goal P_S, the success cost R_S, and the exploration cost R_E — and solving for the order of actions which minimizes the equation, we can solve for the optimal high-level plan in a computationally tractable way. However, each of these terms are implicit expectations over the future belief b_{t+1} and, as before, require great effort and knowledge about the distribution over environments to compute exactly. So, rather than attempting to compute these terms, we endeavor to learn them, allowing us to predict information about actions that enter the unseen regions of the environment.

One point of note is that, since using Dijkstra's algorithm to compute the distance between the goal q_G and each subgoal q_g provides a reasonable estimate of the success cost, we instead learn the *delta success cost*, the difference between the two: $\Delta R_S = R_S(b_t, a_t) - D(b_t, q_g, q_G)$. These terms are learned in a supervised manner: we train a fully-connected neural network to take as input (1) a single sensor observation, a 256-element vector of LIDAR range measurements, (2) the 2D robot-frame position of the goal, and (3) the robot-frame position of the subgoal, and predict the terms specified above. As new observations are incorporated into the map and the set of subgoals changes — i.e., the boundary of observed space changes — any new subgoal is passed to the neural network, which then estimates the properties of the unknown space beyond that subgoal.

The neural network used in this implementation is a fully connected network with hidden layers of width $\{256, 128, 48, 16\}$, each followed a batch normalization layer, 50% dropout, and a sigmoid activation function. We trained our network using the Adam optimizer in TensorFlow (default parameters) over 100k steps with a batch size of 256 and using roughly 300k training samples for each environment. The learning rate begins at 0.004 and decreases by a factor of 0.9 every 5k steps; the relatively fast learning rate decay and the dropout regularize the network and reduce overfitting. The performance is largely insensitive to changes in the random seed, adding additional layers, or changing the number of hidden nodes. We trained a few dozen different networks with variations on these parameters, and found that the standard deviation in the loss between these different networks was only 4.2%. The output of the neural network has three dimensions, for each of the three desired outputs; a sigmoid activation and weighted cross-entropy loss is used to predict the success probability while a linear activation and a squared error loss is used for the two regression outputs. Loss for the success cost is only applied when a plan through a subgoal does lead to the goal and the loss for the exploration cost is only applied when it does not.

One additional consideration is that the effective cost of misclassifying a subgoal strongly depends on a number of factors. For example, if a subgoal constitutes the only path to the goal, then the penalty for misclassifying this subgoal is very high. By contrast, deciding to explore a small room near the goal will not likely take long and is therefore of relatively low cost. Computing the relative importance of correctly classifying a subgoal would require solving the full Bellman Equation and is therefore too expensive to compute. Instead, we provide a heuristic for computing this *misclassification cost* at training time. For subgoals that *do not* lead to the goal, misclassification results in the exploration cost R_E described above, which is roughly the cost of exploring the space beyond the chosen subgoal. For subgoals that *do* lead to the goal, the robot could travel across the entire map before returning; therefore, the penalty is the cost of traveling to the furthest point beyond a subgoal that does not lead to the goal and back. The computed costs are then used as weights for the cross-entropy loss used to train the classifier. Our heuristic matches intuition: subgoals that lead to the goal are naturally more important to correctly classify than those that do not. Finally, we also use *class reweighting* to compensate for the asymmetry in the proportion of subgoals that do and do not lead to the goal.

3.2.3 Training Data Generation

We require data to train our network to learn the properties of unknown space beyond each subgoal. In order to generate this data, we build simulated maps from a few different classes of environment (presented in Section 3.3), and have a point robot navigate from a randomly generated start pose to a randomly chosen goal position. We choose to have the agent assume that all space it has not yet seen is free of obstacles in order for it more fully explore the simulated world to ensure it collects data that will help it learn to recover from dead-ends. As the simulated agent explores, it reveals its environment and generates new boundaries between free and unknown space. We extract the newly updated subgoals at each step, and using the underlying map, we determine if planning through each subgoal will lead to the goal, and its cost of success (if the subgoal *does* lead to the goal) or its exploration cost (if the subgoal *does not* lead to the goal). Each new subgoal corresponds to a single training datum. For each environment, we generate a few hundred maps and collect data from each run of the optimistic planner. The resulting data is then used to train the neural network described in the previous section. We note that the maps we use for training are distinct from those used for testing, though are of the same class of environment. Additionally, for our floor plan environment, the maps we use to generate our training data are from different buildings than the floor plans we use to evaluate performance, so that we may show that we learn generally informative features about human-structured environments instead of memorizing the environments themselves.

3.2.4 Implementation Details

In this implementation of the Learned Subgoal Planning algorithm, the robot builds a 2D occupancy grid as it travels, and from that map extracts the subgoals in order to plan its path. The abstraction of using these high-level actions to plan is what makes this approach computationally tractable. The process of extracting the actions from the grid is as simple as identifying each continuous instance where there exists grid cells marked as "free" adjacent to grid cells that are "unknown." These boundaries mark the complete set of frontiers which the robot can travel through in order to enter unknown space. Since the robot must pass through one of these to reach the goal, the action set that we have defined ensures that by picking one of these actions until the goal is found, the robot is guaranteed to eventually accomplish its task.

It is important to note that if it is not possible to build a reliable occupancy map — for example if the robot was equipped with a monocular camera instead of a laser scanner — then a different method of computing the set of high-level actions would be necessary. This is touched on in Chapter 4. In principle, as long as the robot is capable of extracting a set of actions where, when all actions have been exhausted, the robot is guaranteed to either find its goal or determine the goal cannot be reached, then the Learned Subgoal Planner can be utilized.

Another detail in the implementation of this work relates to how the robot chooses which action to take in the Learned Subgoal Planner. While this has been described above, the planning loop is outlined here for clarity.

1. The robot extracts all frontiers from the environment, and associates a subgoal with each.

2. For each subgoal, the properties highlighted in Figure. 3-1 —the probability of succeeding in reaching the goal P_S, the success cost R_S, the exploration cost R_E — are calculated by the neural network in order to compute the overall cost of taking each individual action.

3. The robot computes its high-level plan by solving the travelling salesman problem associated with determining in which order it should plan to take each action.

4. Once the full high-level plan is found, the robot takes the first action in that list of actions by computing the low-level plan that would take the robot through the subgoal.

5. After this plan is found, the robot travels along this path until it receives a new observation of the world, at which point it updates its map, and starts the planning loop from the beginning again.

Though replanning at each time step might seem computationally expensive, there are a few ways this process can be accelerated. First, the properties of a frontier only change when it has been observed in a given time step. Thus, only the frontiers near the robot that have been seen are updated by the neural network. Second, it was found experimentally that certain actions can be ignored without much effect in the planning loop if they have a low enough probability of success. So, in both simulated and real trials, it is wise to limit the robot to consider only the n most likely subgoals

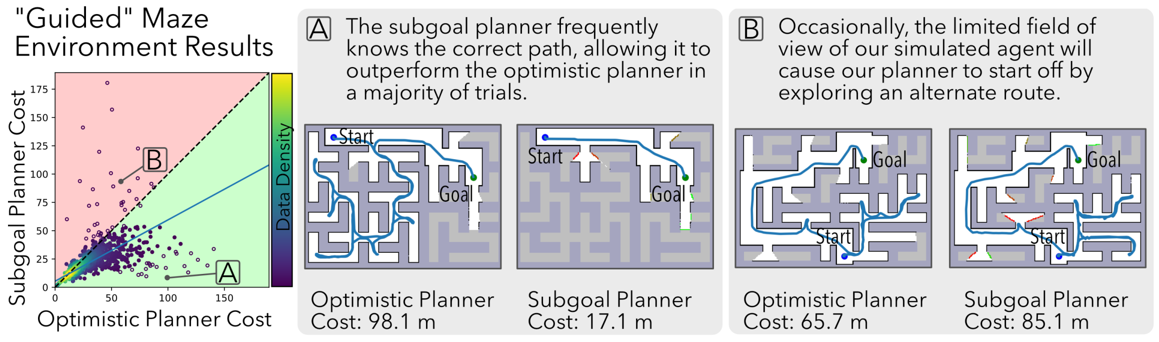

Figure 3-2: A comparison between the cost-to-go of the optimistic planner baseline and our subgoal planner for 1,200 simulated trials in the Guided Maze Environment. The blue line represents the calculated fit line of data points selected by RANSAC (filled circles). The Learned Subgoal Planning agent travels a total of 22.3% less distance than the optimistic baseline over these trials, which is shown in the scatter plot by the fit line being in the green colored region. In plots of the subgoal planner's path, boundaries associated with each subgoal are colored from green to red so as to visualize P_S, the estimated likelihood that it will lead to the goal.

when it is on the third step of the planning loop described above. How n is chosen depends on the environment type and the available computation. These optimizations allowed us to run the subgoal planning algorithm at approximately 2-Hz on the real robot, the specs for which are described in Section 3.4.

3.3 Simulation Results

To evaluate the performance of our approach to planning, we generate simulated occupancy grids from four different classes of environments: *"guided" mazes*, *random forests*, *office-like environments*, and *real-world floor plans*. From these samples of these environments, we generate training data and train a neural network for each class, as described in Sec. 3.2.2 & 3.2.3. In each environment, individual trials consist of our simulated RC Car agent navigating between a random start/goal location for each of the optimistic planner — which plans as if all unknown space is unoccupied — and Learned Subgoal Planner. For each environment we test in, we compare the length of the resulting trajectories to determine how the Learned Subgoal Planner performed relative to the baseline.

3.3.1 "Guided" Maze Navigation Results

We generate maze environments using *Kruskal's Algorithm* [1] as a proof of concept for the Learned Subgoal Planning Algorithm. We use the term *guided* to mean that the maze is designed in such a way that local features give immediate guidance on which way to go. Specifically, this means the hallways along the most-direct path between the start and the goal are twice as wide as any other branches of the maze. With this clear marker, we would expect our planner to be able to learn that the expected cost to reach the goal will likely be much lower for an action that stays to the wider path. Because this information is in the geometry of the environment, individual laser scans should be enough to estimate properties of the unknown space beyond each subgoal with reasonable reliability. We evaluate navigation performance of the simulated agents through 1,200 runs in our synthetic maze environment, and include a scatter plot in Fig. 3-2 that gives the net distance traveled of each trial by both the optimistic planner baseline and our subgoal planner.

Evaluating the results of our trials in the *"guided" mazes*, we find the net cost incurred by the robot summed over all trials (each from start to goal) shown in Fig. 3-2 is 22.3% lower for our subgoal planner than for the optimistic planner. Furthermore, fitting a line to 95% of the data — as selected by RANSAC — compares the incurred cost of our planner to that of the optimistic planner, yielding a cost ratio of 0.570: the expected cost of the motion plan executed by our subgoal planner is 43% lower than that of the optimistic planner for the same environment configuration. Fig. 3-2A shows one example where the subgoal planner, using local features to estimate the expected cost of each subgoal, navigates directly to the goal, yet the optimistic planner unnecessarily explores much of the environment before eventually reaching the goal. By learning to stick to the wide hallways, our planner has a distinct advantage over one which assumes all paths through unknown space lead directly to the goal.

There are a handful of cases however in which the optimistic planner outperforms the Learned Subgoal Planner. Examples of these cases are typically a result of incorrect predictions from the network near the start of a plan. This is potentially because

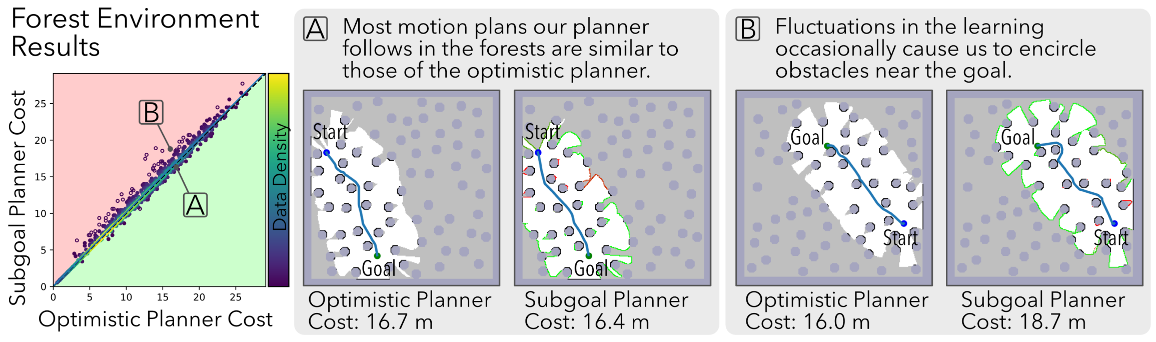

Figure 3-3: A comparison between the cost-to-go of the optimistic planner baseline and our Learned Subgoal Planner for 1,000 simulated trials in the Forest Environment. Our agent closely matches the performance of the optimistic baseline, which already achieves near-optimal navigation.

it can be difficult to determine which path is wider in certain starting positions. If our planner incorrectly predicts that the subgoal leading to the goal is a dead end, it will navigate a portion of the remaining environment before returning to correct its mistake. We show one such example in Fig. 3-2B. These cases are uncommon, however, (roughly 2.5% of results) and are outweighed by the trials in which our planner outperforms the optimistic planner.

3.3.2 Forest Navigation Results

Having demonstrated that our algorithm is capable of using local features to aid navigation, it is also important to show that the Learned Subgoal Planner does not lower performance in the case were there is not much information to be gained from local geometry. If the world were completely random, the optimal policy might be to enter unknown space in the region closest to the goal, which is approximately the policy of the optimistic planner. To show that the Learned Subgoal Planner can navigate in such environments, we have our simulated agent navigate in a set of random forests, generated in such a way that gaps between each individual cylinder is large enough to allow the agent to pass between them. We conducted 1,000 trials in the random forest environments and plot the results in Fig. 3-3, which shows a good agreement between the cost of the motion plans from the optimistic planner and our subgoal planner. Fitting a line to 95% of the data — as selected by RANSAC — yields a cost ratio of 1.006 between the planners, confirming the good agreement.

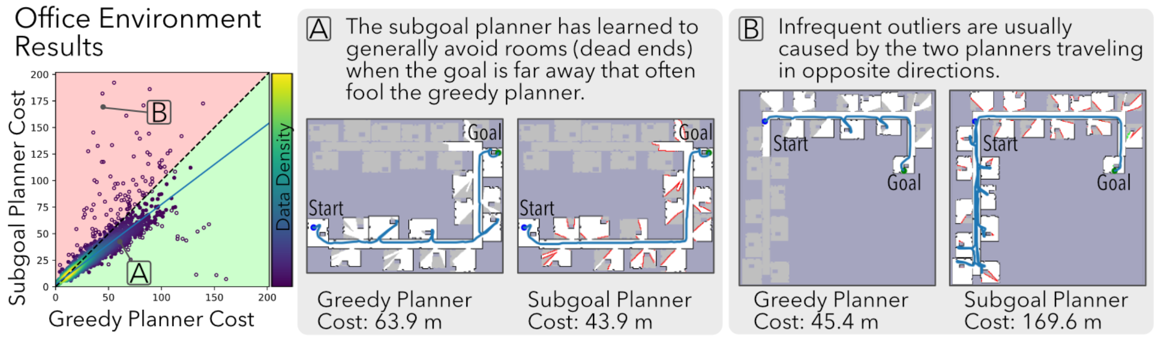

Figure 3-4: A comparison between the cost-to-go of our Learned Subgoal Planner and the optimistic planner baseline for 2,500 simulated trials in the Office-Like environment.

In addition, the cost of using our subgoal planner summed over all trials is only 2% higher than the net cost from the optimistic planner, suggesting that using our subgoal planner does not substantively lower performance in environments for which the optimistic planner is already very effective.

It is worth noting that outlying points, those visibly above the line, are uncommon and have only a small impact on the overall navigation performance; such points, such as the one shown in Fig. 3-3B, frequently occur near the goal, when small fluctuations in the learning output may cause our planner to change directions while encircling a cylindrical object. There are surprisingly a few cases where the Learned Subgoal Planner outperforms the optimistic planner in this environment type. As shown in Fig. 3-3A, such instances are caused mostly by luck, when the two motion plans deviate around an obstacle which was occluding another.

3.3.3 Office-Like Environment Navigation Results

The previous environments were illustrative of two important properties of the Learned Subgoal Planner. The *guided maze* environment demonstrated that the deep network was able to learn to associate useful structural information with the higher level actions, while the *random forest* showed the algorithm was able to efficiently navigate to the goal even in the worst case scenario when the was no usable information in the structure to be leveraged. However, neither of these were particularly realistic environments that a robot might face when solving a navigation task. In order to

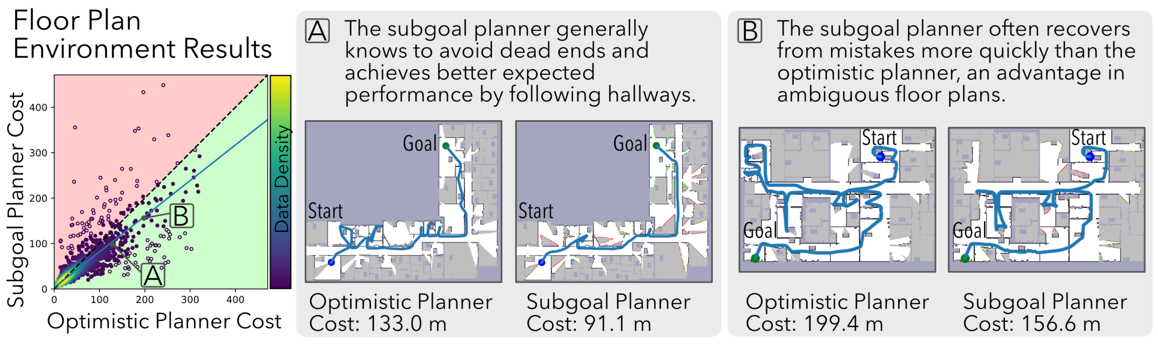

Figure 3-5: A comparison between the cost-to-go of our Learned Subgoal Planner and the optimistic planner baseline for 2,000 simulated trials in the Floor Plan Environment. The subgoal planning agent travels a total of 12.5% less distance than the optimistic baseline.

demonstrate that the Learned Subgoal Planner approach was able to navigate environments that a robot might actually see in the real wold, we first hand-crafted an *offlice-like* environment that mimics what a human-structured indoor environment looks like.

In Fig. 3-4 we see the results of 2,500 simulated trials in these *office-like* environments. The environment is structured as one long hallway, with small offices attached. Within each office is an obstacle (similar to a desk) placed such that the robot cannot reveal the full space without entering each room. Given enough data, we would expect the robot to learn not to enter the small rooms until it is near the goal, and that is exactly what we saw in the results. Cases where the optimistic planner outperforms the Learned Subgoal Planner were generally due to cases where the robot had to guess going either left or right down a hallway. In instances where it is not clear, we would expect the robot to occasionally choose the "wrong" direction. As shown in the scatter plot however, in expectation, the learned subgoal planning approach well outperforms the optimistic planner.

3.3.4 Floor Plan Environment Navigation Results

Finally, we conducted trials in the *floor plan environment*: maps of our floor plan environment are occupancy grids generated from blueprints of real-world buildings around the MIT campus with obstacles added at random, so as to mimic real-world furniture and clutter. A scatter plot showing the results of these trials can be found in Fig. 3-5. Our Learned Subgoal Planner achieves better performance in expectation for the floor plan environments, which is impressive due to the complex nature of some of the maps. The net cost over all runs is 12.5% lower for the subgoal planner than for the optimistic planner. We again fit a line to 95% of the data, as selected by RANSAC; the cost ratio is 0.784, suggesting that the expected cost of a plan using the Learned Subgoal Planner is 21% shorter than that of the optimistic planner.

In Fig. 3-5A, we show a typical sample from the navigation trials, in which our Learned Subgoal Planner avoids many dead ends — which take the form of offices and lab spaces — that lead the optimistic planner astray. In most floor plan maps, hallways define the most likely routes between faraway start and goal locations; thus, subgoals that lead down hallways are frequently of lowest expected cost. As with the *office-like* environments, most outliers in Fig. 3-5 occur when the two planners travel in different directions at a branching point in the environment.

What is particularly interesting about these environments based on the real world is that the fastest route to the goal is not always found by sticking to hallways. Because of this, unlike the *office-like* environments, we found that the learned planner will occasionally enter a room far from the goal if it can see from outside that a shortcut might exist. Relatedly, we highlight in Fig. 3-5B one case in which following the hallway does not lead to the goal. After traveling much of the hallway, the subgoal planning agent recovers and turns back to seek more probable route to the goal. It reaches the goal 22% faster than the the optimistic agent, which goes on to explore the upper-left portion of the map.

3.4 Real-world Experiments

Finally, we integrated our Learned Subgoal Planner with a physical platform so as to observe its performance in the real world. For our experimental platform, we used a RC car [49] with a Hokuyo LIDAR [25] and a Microstrain IMU [40]. Several modifications were made to the robot, including replacing the suspension system with a rigid support to prevent the LIDAR from changing its angle as the robot acceler-

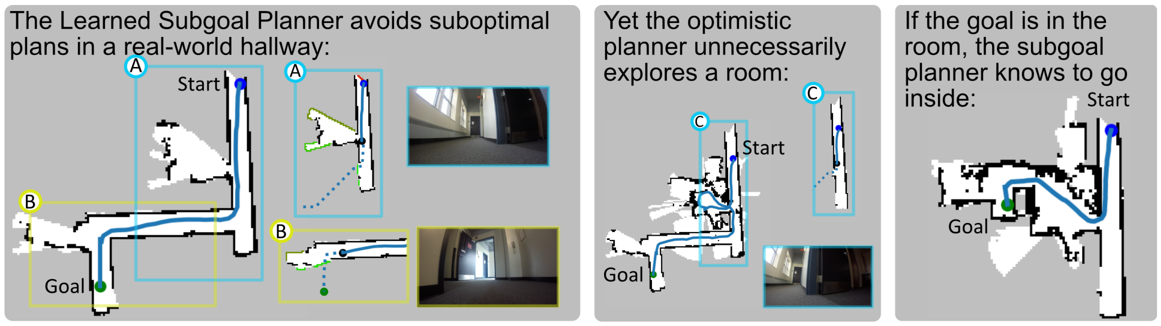
The Learned Subgoal Planner avoids suboptimal plans in a real-world hallway:
A
Start
A
B
B
Goal
Yet the optimistic planner unnecessarily explores a room:
C
C
Start
Goal
If the goal is in the room, the subgoal planner knows to go inside:
Start
Goal

Figure 3-6: This figure shows an experimental run on our RC Car, in which our Learned Subgoal Planner (left) outperforms the optimistic planner (center) in navigating to an unseen goal. In (A), we observe the subgoal planner choose to pass the room and continue down the hallway. By contrast, the optimistic planner enters the room (C). On the far right, our Learned Subgoal Planner knows to enter the room when the goal is placed in a position where the lowest expected cost path is to go inside.

ated. Additionally, a camera was mounted to capture images from the perspective of the robot. The Robot Operating System was used for interprocess communication, run on a NUC computer [27]. The Octomap package [26] was used to build a 2D occupancy grid from the incoming laser scans and the robot pose, provided by an Extended Kalman Filter that fuses the output from a laser scan matcher and IMU measurements. The planning process on the car was almost exactly same as in simulation. The primary difference was, once a subgoal had been selected, a pure pursuit controller [10] was used to determine the robot's motion primitive. Rather than collect a large-volume of real-world data, we instead trained in simulated floorplan environments with a realistic distribution of open doorways. Using this data was a natural choice, since the real-world environments we use for testing are similar in structure to the floor plans.

We focused on locations known to cause problems for optimistic approaches to navigating unknown spaces, particularly hallways with classrooms or other rooms. As in our simulation results, agents using an optimistic planner frequently entered rooms, even when the goal was sufficiently far away that reaching the goal via entering the room was unlikely. In three distinct, real-world environments, we observed similar behavior: the subgoal planning agent would actively avoid unnecessarily entering dead-ends — including classrooms, a lecture hall, an apartment, and numerous closets — which the optimistic planner would enter. Yet if the goal was placed inside these regions, the subgoal planner would know to enter them, showing that the agent knew more than to simply follow hallway-like features. We highlight one of our real-world experiments in Fig. 3-6, in which the agent demonstrates the aforementioned behavior in an apartment complex. By avoiding the open rooms and dead-ends, the agent running our planner reached faraway goals more quickly than the optimistic planner, thus corroborating our simulation results with a real-world platform.

One impact of implementing the Learned Subgoal Planner using an occupancy grid was that, because the input to the network was range data from a laser scanner, the network that was trained on simulated data in the MIT floorplan environments was able to generalize to the real world. Though the range measurements used in the simulator likely had different noise characteristics than the LIDAR used on the robot, the discrepancies were not significant enough to affect the output of the neural network. This made testing the system on the robot far easier than it would have been otherwise, as it was not necessary to collect new data on the robot where we did not always have access to a ground-truth map. Work in the next chapter will discuss incorporating visual input into the Learned Subgoal Planning approach, which while a valuable source of useful information for the robot, makes this once seamless transfer significantly less so.

3.5 Discussion

In this chapter, the Learned Subgoal Planner was presented as an approach to use subgoals, each corresponding to a boundary between free and unknown space, to reduce the complexity of navigating unknown environments. An accompanying factorization of the Bellman Equation was also given for evaluating the expected cost of navigating through unknown space to reach an unseen goal in partially-revealed environments by learning properties related to the subgoals. Each subgoal represents an action the robot might take to explore unknown space, and, since these are the full set of actions available to the robot, we are guaranteed to eventually reach the goal. With this planning framework, we can incorporate prior information into our decision making process in a way that deterministically reaches the goal and is computationally tractable. We show that our Learned Subgoal Planner is capable of making informed decisions in both simulated environments and on real-world hardware. One potential avenue to improve our work is augmenting the learning pipeline to use more than a single sensor observation, which could allow the system to update the properties of faraway subgoals as the robot explores.

This representation is also not without limitations. We make an independence assumption between our actions so that our model does not capture the impact of information gathering actions. Second, our neural network model returns only the maximum likelihood value for each learned subgoal property: our model may have trouble when higher-order moments of the distribution matter a great deal. Despite these limitations, our planner demonstrates reductions of over 20% in the expected cost-to-go in simulated floor plan environments. Furthermore, we demonstrate promising results on a real-world platform, showing that our Learned Subgoal Planner is a practical method for improving autonomous navigation of structured unknown environments.

Another way this work could be improved would be to include visual input into the learning pipeline. Image a robot tasked with exiting a building. If the agent were able to identify markers such as exit signs above doors, the task would be much easier than if it were forced to rely only on the geometric input provided by a laser scan. In fact, if we were able to replace the laser scanner entirely with a monocular camera, it would be much more reasonable to extend this work to platforms (such as quad-rotors) where laser scanners are often imperfect sensors due to their size and weight.

In order to make this leap however, the occupancy grid, which is relied upon to both build a map of explored space and extract the available subgoals for high-level planning, would need to be replaced with a system that can replicate those properties from visual input. In the next chapter, we investigate a different map representation, generated from visual input, that allows for a robot to utilize the Learned Subgoal Planner by directly identifying actions from a panoramic monocular image. Though the extension of incorporating this map with the planner is left for future work, with this framework we hope to be able to plan with the Learned Subgoal Planner without relying upon specialized hardware, using a sparse map representation.

Chapter 4

Mapping and Data-Association with High-Level Actions

The work presented in Chapter 3, as discussed in the introduction to this thesis, addresses one of the two primary challenges in the problem of the autonomous navigation of unknown environments. By reasoning between which action to take to enter unknown space, a robot using the Learned Subgoal Planner is capable of intelligently planning to navigate through unknown regions of its world. Also necessary however, is the ability to build a map of the environment as the robot travels. In Chapter 3, the map used was a 2D occupancy grid, though this brings with it a few limitations, particularly that it can be difficult to reliably construct such a map from monocular visual input (Section 4.1). Moreover, planning through such an environment can be complex as the size of the map grows. In this chapter, we investigate a novel map representation which directly identifies the actions available to a robot at any position by detecting high level actions from a panoramic image. From these detections we build a sparse map of high-level actions to use for planning. A major focus of this chapter addresses the data-association problem for these detected actions.

4.1 Constructing a Map of Actions

The Learned Subgoal Planner approaches navigation in partially explored environments by considering entering unknown space through different frontiers as explicitly different high-level actions in order to efficiently reason about the implications of taking one of these actions. Implicit in this approach is the ability to extract higher-level actions from the map representation of the environment in a way that guarantees all possible options are considered. For simple cases where the only actions that are available to the robot are entering unknown space in a 2D occupancy grid, this problem can be solved by manipulating the map after it has been built. However, as the action space becomes more complex, or the sensors available to build a map change, how to decide what constitutes an action becomes less clear.

The field of mobile robotics has become increasingly interested in systems that operate with a single monocular camera for a multitude of reasons, not the least of which is that weight and cost considerations often make mounting a laser scanner atop a mobile robot such as a quad-rotor impractical [25]. Common approaches to vision-based map construction and navigation rely on storing many sparse landmarks to represent the geometry, making reasoning about false-positive and false-negative detections computationally intractable for non-trivial problems [45]. Furthermore, most such representations unnecessarily include elements like small clutter or dynamic objects that complicate the map building process, and which a local reactive planner is well-suited to avoid.

Maps produced by modern monocular SLAM systems are designed to be used for localization, and are not inherently useful for navigation. Consider, for example, the map output by a modern feature based SLAM system such as ORB-SLAM [45]. While sparse visual features are well-suited for the task of building a map for localization, reasoning about higher-level actions in these maps is difficult, if not impossible. The reason for this is that, unlike an occupancy grid, point-clouds of visual features do not distinguish between known and unknown space directly, and therefore, it is not feasible to estimate the properties of plans which enter unknown space in a POMDP framework, as was presented in chapter 3.

The difficulty of using point-clouds for navigation is that they are not intended to represent the structure of an environment. No matter how the map is manipulated, there will always be a risk that the true topology of the environment is not captured, because that is not what these features are designed to do. It is possible of course, from a map of sparse landmarks, to discretize the space, and consider regions that contain a certain number of landmarks to be obstacles in order to construct an estimate of space that contains an obstacle. Combining the projection with the complete history of robot poses would, in theory, give the ability to construct an occupancy grid from a point-cloud. Unfortunately however, monocular sensing is imperfect and noisy measurements may occlude critical routes to the goal or introduce non-existent free space that can lead an agent astray or, worse, mark as visited unseen space that may have contained the goal. Even if the detector extracts features well, there may be regions of the environment, such as a blank wall, where there are simply not enough features to support such a system reliably.

Instead of trying to build a dense map that represents the structure of the environment perfectly, then extracting high-level actions from the map (as in chapter 3), the work in Chapter 4 first discusses a novel method to build a map from the high-level actions directly, represented as a graph. To construct the map, the underlying structure of obstacles in the environment is inferred using a Convolutional Neural Network (CNN) to detect polygonal corners from panoramic images and estimate an additional property that represents whether or not occluding structure exists between each landmark. By building this structure, and storing a history of past robot poses, we are able to track what regions of space have been observed, and thus can know which regions of space have never been seen. With the action based map representation, as well as a reactive planner to help avoid obstacles not included in the map, the robot is capable of reasoning about the high-level actions available to it for navigation directly at any time.

Critical in building any map on a real robot is the ability to be robust to noise, both in the detection of landmarks, as well as the the data-association problem. In this chapter, the map building process with a sparse action detector is described first for noiseless detections (Section 4.2). Then, the case of an imperfect sensor is discussed, and a method for eliminating outlying detections by using information about occluded space is presented (Section 4.3). Once the map building process has been introduced, special attention is paid to the process of *data-association*, which is a primary focus of this work (Section 4.4). Two different metrics for data-association are explored, one of which is a novel method that incorporates visual information using convolutional neural networks. These techniques are compared using simulated, noisy data.

4.2 Map-Building

Before discussing how we build a map of actions from a noisy, learned, vision-based sensor, we first present how we construct our map representation using perfect sensing. Here, we introduce key terms and show that our map is sufficient for navigation of unknown environments.

4.2.1 Noiseless Map Building

To build our map, a robot is equipped with a sensor that can detect the presence and form of obstacles by decomposing an image into features that correspond to locally visible actions (e.g. corners or vertices) and the occluding structure that connects them (e.g. the presence and geometry of walls). The reason we have chosen this form is that, in the context of navigating two dimensional environments, the actions available to a robot are to travel around a corner in order to reveal the unknown space beyond it. Thus, if we are able to construct a map of polygonal vertices, plus information about whether it is possible to travel in-between these vertices, we will have a map representation where the robot will know exactly what actions are available to it at all times.

The actual detection of both the vertices and their connecting walls is accomplished with a convolutional neural network, which will be discussed in chapter 4.2.2. The output of this network is an observation z, which contains an ordered list of these

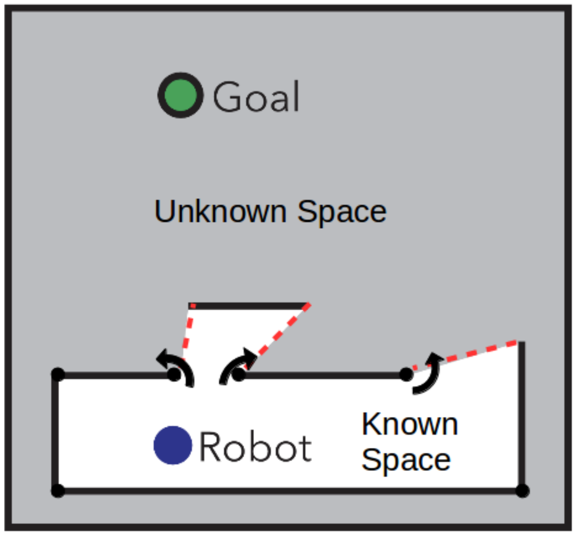

Figure 4-1: This figure depicts the star shaped region of local free space visible to a robot detecting vertices and the structure connecting them. Vertices (black dots) that lie on the boundary of that space (red dashed lines) constitute the actions available to the robot.

vertex detections $\sigma_v \in \mathbb{R}^2$ and whether or not occluding structure connects pairs of co-visible vertices σ_w. In the ideal case, our sensor is noise-free and detections of σ_w alone can be used to define a *star-shaped* region of local free space $s(z)$ originating from the robot (See Fig. 4-1). In totality, the union of the regions observed by the robot as it navigates forms the set of observed space $S = \bigcup_{i=0}^{N_{\text{obs}}} s(z^{(i)})$. We can define an action as the robot rounding any corner that exists on the boundary of S.

Navigating the environment involves revealing unknown space, either by sight or by traversal, which can be accomplished by navigating to the *frontier* — the traversable boundary of S — defined as $F = \partial S - \bigcup \sigma_o$, where ∂S is the boundary of S. We represent the map state m as a graph, where the nodes in the graph V encode information about the set of vertices and the edges of the graph W encode the relationships between vertices as defined by the set of occluding structure between vertices. It is important to note that this information can be used to determine whether an action can be taken from the robot's current position. If a vertex has structure between it and its neighboring vertex — given by σ_w — it is not possible to enter unknown space by going between those two vertices. When there is no such occluding structure between vertices, there is a local frontier between them. From the map m and observed space S, the frontier F can be computed. Our planner

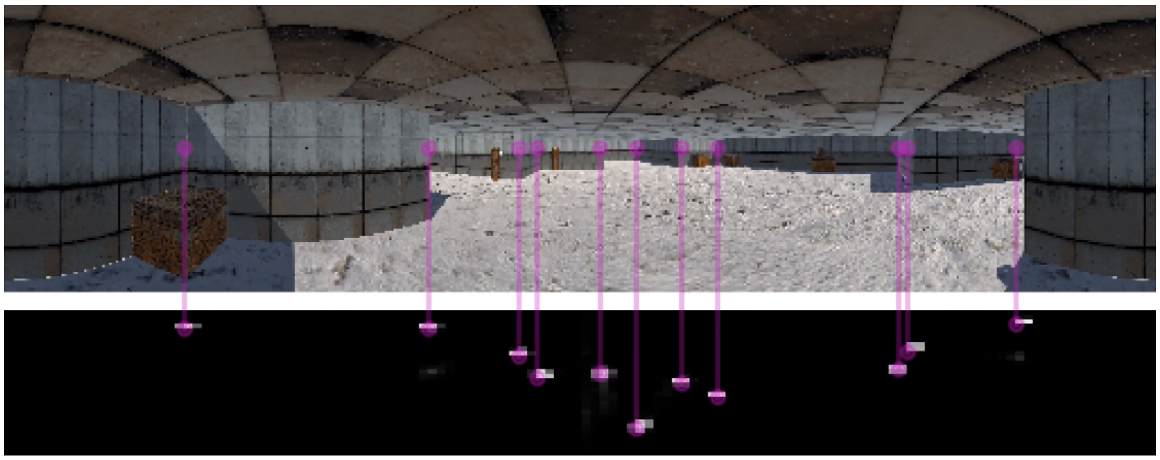

Figure 4-2: Here we see the output of the neural network in a simulated indoor environment. In addition to estimating the position at which a vertex exisits relative to the robot, the network also estimates the orientation (classification) of each vertex, which are then used to construct the map.

explores or navigates to an unseen goal by choosing to take an action on the frontier (rounding a corner that exists on the frontier), thus traversing it. In so doing, we are guaranteed to advance the frontier, which makes our map representation sufficient for goal directed navigation through unexplored environments. In a task like the one presented in Chapter 3, navigation terminates either when the goal is reached or when the frontier is the null set, indicating we have completely explored the environment.

4.2.2 Detecting Actions with Convolutional Neural Networks

In the case of perfect sensing, we imagined the robot to be equipped with an action sensor such that each observation z contains a list of vertex detections σ_v, ordered by the angle at which they were observed. Instead of relying on some imagined sensor, we endeavor to learn how to detect vertices and their properties from a panoramic image in simulated environments. To do this, we must first define the simulated environment in which our robot will navigate. We generated hundreds of simulated indoor environments, each composed of a small labyrinth of rooms connected by corridors, using a Unity Game and Physics Engine. To collect data, we simulated the traversal of these worlds by a point-robot equipped with a panoramic camera. An image from one instance of these environments, taken from the perspective of the robot, can be seen in figure 4-2.

Attempting to directly learn whether a wall exists between vertices (σ_w) is challenging, so instead each detected vertex is classified according to whether or not it connects to its neighbors (in angle order). For each observation, the 4-class classification — which we refer to as *vertex orientation* — of neighboring vertices can be combined to estimate the likelihood a wall exists between them. Each vertex detection therefore consists of a *position* with hand-coded *covariance* and the *vertex orientation*, which is a point on a 4D simplex. The corners of the simplex represent if the vertex only connects to the vertex on its right, only connects to the vertex on its left, connects to both, or connects to neither. The sensor itself is assumed to be fundamentally noisy, and each observation may have extraneous landmark detections (false-positive detections) or missing detections (false-negative detections) each with an approximately known rate of occurrence.

We train a convolutional neural network (CNN) to estimate the likelihood of a vertex existing in a particular position relative to the robot, along with a few other properties: the 4D estimated vertex orientation (classification). The network takes a 128×512 pixel RGB panoramic image as input and outputs the likelihood a vertex exists at a point in a 32×128 output representing relative range and bearing from the robot. For each point in the output, the network also predicts the estimated vertex orientation at the same resolution . The network begins with 5 convolutional layers — of kernel dimension ([3,3,3,64], [3,3,64,64], [3,3,64,128], [3,3,128,128], [3,3,128,256]) — each executed twice, and followed by batch normalization and a max-pool layer. Following the convolutional encoder layers, we add 3 convolutional decoder layers — of kernel dimension ([3,3,256,128], [3,3,128,64], [3,3,64,64]) — followed again by batch normilization and max-pooling. The output of this decoder portion is then of dimension [32, 128, 5], and is fed to two different output layers (each fully connected) with different activation functions: a sigmoid output for the vertex likelihood, and a 4-way softmax output the four matrices of the vertex orientation. Fig. 4-2 shows the output of the vertex detection part of the neural network trained on a simulated hallway environment. The output of the network includes the *vertex likelihood*, on which we use peak detection to convert the measurement to discrete vertex observations and

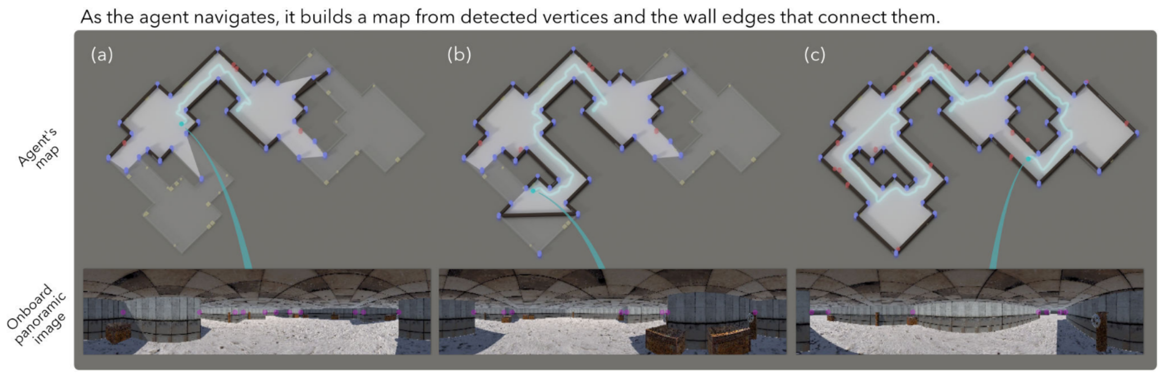
As the agent navigates, it builds a map from detected vertices and the wall edges that connect them.
(a)
(b)
(c)
Agent's map
Onboard panoramic image

Figure 4-3: This figure shows how we construct a map online. The first row is a visualization of the best estimate for the map at each frame. Black lines represent wall edges, light blue points represent vertices, and red points represent vertices that have been removed from the graph via sampling. To remove these erroneous detections, we sample over the map domain, enabling and disabling potential walls and vertices, to determine the most likely map as the robot travels (see Sec. 4.3.1). The estimate of known space is lighter grey, while the darker surrounding is unexplored. Yellow points in the image represent clutter in the world, which the vertex detector is capable of ignoring The second row is the panoramic point of view of the robot in each frame, where purple circles show vertex detections. By the end of travel, the estimated map closely matches the underlying geometry; erroneous vertex detections do not appear.

covariances. In addition to this output is a four-channel matrix of the same dimension corresponding to the vertex orientation. Though the network does occasionally suffer from false-positive and false-negative detections, our mapping procedure (Sec. 4.3) is designed with a noisy sensor in mind, and is robust to occasional errors like these.

4.2.3 Noisy Map Building Procedure

Environments that we want robots to traverse are complex, and mapping is necessary to track high-level structure. In practice, our learned vertex detector will unavoidably produce noise. This requires a probabilistic model so that the most likely map can be estimated from the set of observations.

As our robot travels through the world, at each time-step it obtains a new observation from the neural network. The output of each observation is a set of *vertex detections* which must be used to build a map of vertices and walls in order to represent the environment. Here, we will define the process by which that happens in our framework.

1. First, the robot receives an observation z, an ordered list of vertices σ_v and

whether or not occluding structure exists between each vertex and the vertex beside it σ_w.

2. Each vertex in this observation is compared to the vertices that exist in the map in order to determine how each detection should be associated. This process is explained in greater detain in section 4.4. The result of this is that either a vertex is matched with an existing one, or it was selected to be a new vertex that must be added to the map.

3. If a vertex is unmatched, it is added to the map, represented by a position in 2D, and a covariance that indicates the uncertainty about that pose. For each vertex detection that was associated with an existing one, the position and covariance are updated via a Kalman Filter.

4. It is at this point in our map building process where we must account for errors in the system. After data-association, we consider removing or adding vertices to the map, and comparing a certain number of maps to find the one that is the most likely given our sensor observations. How this is done (and how we determine where the walls in a map are) is presented in section 4.3.

5. Once the most likely map has been found, the robot can identify which action it wants to take by selecting a vertex that is on the boundary between free and unknown space. If this vertex is not directly visible, it must plan a path through the map by connecting co-visible vertices. With each step along a planned path, the map building process is repeated.

4.3 Map Estimation with Noisy Sensing

The output of each observation is a set of *vertex detections* which, by the process discussed in Section 4.4, will be associated with one of the vertices that exist in the map. Due to noise in the detector, some of these vertices may be false-positive detections. Similarly, the detector may have failed to recognize a vertex, marking a false-negative detection. Perhaps most significantly, errors in the estimated position of a *vertex detection* might cause it to be associated with the wrong vertex in the map. These compounding sources of noise may lead to a map where there exist vertices which do not belong.

In order to overcome these sources of potential error, we recognize that only a subset of the vertices which have been detected over the life of the robot should appear in the final map. The set of all vertices defines the space of possible maps, where a proposed map is an inclusive subset of this and a set of wall edges that connect pairs vertices from the selected subset. We sample over this space in order to determine the most likely arrangement of vertices and walls based on the robot's history of observations. In particular, each sampling step we randomly select a potential vertex and either add or remove it from the set of selected vertices. Note that a vertex can only be added back to the map if it were previously removed. By turning vertices on and off in this way, we can create a set of maps to compare. In order to determine which of these is the most likely however, it is necessary to define a posterior based on observations of the world the robot has seen over past poses.

4.3.1 Defining the posterior

The posterior over maps can be expanded as follows using Bayes Rule:

$$\log P(m|z^1, \cdots, z^N) = \sum_{i=0}^{N} \log P(z^i|m) + \log P(m) - \log P(z^1, \cdots, z^N) \tag{4.1}$$

where z represents an observation, m represents a map, and we have assumed that the sensor observations are i.i.d to factor $\log P(z^1, \cdots, z^N|m)$. The final term, the likelihood of a sensor measurement, is a constant and can be ignored. The prior distribution over maps, $P(m)$, is intractably difficult to obtain in practice. Yet we can use this term to incorporate some global priors about the map, namely biases against the addition of new vertices and walls, which helps to eliminate false-positive landmarks.

The likelihood of a sensor observation $P(z^i|m)$ is at the heart of the posterior distribution. To compute this likelihood, we generate a *hypothetical observation* z_h from the robot's position at time i, x^i, and compare it to the real observation z^i. A hypothetical observation is generated by ray-casting against the proposed map, and determining what the robot *would have seen* had it received a perfect observation in that map at that pose. Any points that appear in the real observation that are not in the expected vertex detection set are *false-positive detections*. Conversely, vertices that appear in the expected vertex detection set but were not in the real observation are *false-negative detections*, since it is expected that the robot should have seen them.

We separately reason about each output of the sensor by splitting the sensor likelihood into two components: (1) the *vertex likelihood* and (2) the *wall likelihood*, and evaluate them. The vertex likelihood depends only on the number of false-positive/negative detections: $P_v = R_{\text{fp}}^{N_{fp}} R_{\text{fn}}^{N_{fn}}$, where R_{fp}, R_{fn} are the rates of false-positive/negative detections and N_{fp}, N_{fn} are the number of false-positive/negative detections.

The wall likelihood uses the real and hypothesized observations to estimate the likelihood that either walls or free space connect vertices. The vertex orientation is a direct measure of structure surrounding a vertex and so the likelihood a wall or free space exists between a pair of vertices depends on the vertex orientation of each. The likelihood of a free space edge w_{lr} existing between two neighboring vertices (the left l and right r vertex observations σ_l and σ_r) depends on their respective orientations (l_l and l_r) and is given by:

$$P(w_{lr}|z) = (1 - P(l_l = \text{no rightward wall}))(1 - P(l_r = \text{no leftward wall})) \qquad (4.2)$$

Each hypothesized vertex observation is paired with the vertex orientation from its associated counterpart; hypothetical detections without an associated real vertex detection (false-negatives) are assigned a value of $[0.25, 0.25, 0.25, 0.25]$. We loop through all pairs of consecutive (in angle) vertex detection and accumulate a factor of $P(w_{lr}|z)$ if a wall exists between the two vertices in the proposed map and a factor of $1 - P(w_{lr}|z)$ if not.

As stated in equation (4.1), in order to evaluate the posterior for a potential map, we evaluate the probability of an observation for a given map at each pose the robot received an observation for. In other words, we compare what the robot did see to what it should have seen for a proposed map at all time steps. By summing the log likelihood for each observation, we can determine the probability of the current map to compare to other proposals.

Our posterior codifies a number of behaviors we would expect to see during our map-building process. First, the posterior allows us to rule out false-positive vertices in a principled manner. Since false-positive vertices can result in hypothetical false-negative observations in the calculation of the posterior, the vertex is ultimately less likely to appear in the underlying geometry. Second, the existence of a wall depends on both its ability to occlude vertices *and* the impact it has on the orientation (class) of the hypothetical vertices to which it connects. By occluding portions of the map, the walls also influence the number of false-positive and false-negative detections recorded during the evaluation of map likelihood. The occlusions allow us to reason about which vertices should be compared to at each robot pose.

4.4 Data-Association

The posterior over maps gives us the ability to determine which of two maps is more likely given a set of sensor observations. However, due to the high dimensionality of the space of possible maps, and the fact that calculating the posterior can be computationally expensive, it is not feasible to consider each possible map that might exist from the observations of our noisy sensor. Thus, in order to find the best possible map in a reasonable number of samples of the posterior (Eq. 4.1), it is important to begin sampling from a map that is near the optimal.

Beginning with an initial map estimate, as new observations are added online, we must use some metric to associate detections. Successful associations of a detected vertex with a vertex in the graph are ultimately used to update the position and covariance of the existing vertex by using a Kalman Filter. Detections which are unassociated with an existing vertex create new vertices in the graph, under the constraint that no two detections from a single observation can be associated with a single existing vertex. The following sections present different metrics for solving the data-association problem.

4.4.1 Mahalanobis Distance for Data-Association

One straightforward technique for data-association takes advantage of how the vertices are represented in the map. Each vertex is represented in the map not only as an x, y position in two dimensions, but also includes a covariance, which is a measure of the confidence in that position. Similarly, each vertex detection, as output from the learned detector, has both a position and covariance estimate as well. Instead of simply comparing the Euclidean distance between a vertex and a vertex detection, we can incorporate more information by including the covariance of the detection in the metric.

The Mahalanobis distance, a measure of distance between a point x and a distribution with mean μ and covariance S, is a metric for determining how many standard deviations away from the mean of a distribution a point is.

$$D_m = \sqrt{(x-\mu)^T S^{-1}(x-\mu)} \tag{4.3}$$

By using the Mahalanobis distance as our metric, we can account for the fact that our vertex detector is, in general, more confident about the position of a vertex in bearing than it is in distance. A vertex that is detected to be five meters away from another might only be one standard deviation away from if we define our covariance to appropriately capture the noise in the range estimate.

Using the Mahalanobis distance as a metric, as new vertex detections are generated, each new detection is associated with an existing vertex if the vertex is within some defined threshold (often one standard deviation of the detection's estimated covariance). Note that we use the covariance of the detection and not the covariance of the existing vertex for Mahalanobis distance, because as more observations are associated with a vertex, the confidence in its position generally increases. Thus, its covariance shrinks, and it becomes more difficult for a detection to be within the defined threshold. It would be incorrect to have a detection be less likely to be associated with a vertex that has been seen many times (in fact the reality is that a vertex that has been seen many times should be more likely to be seen again). Thus, it is the covariance of the detected vertex that is used to determine the Mahalanobis distance.

An interesting result of using Mahalanobis distance for data-association is that multiple vertex detections might be within the threshold of association of a single vertex, and vice versa. In this case, we take advantage of the concept of mutual exclusion [42], which stipulates that each landmark can only be associated with one detection per observation. In order to ensure that the optimal associations are found, even in the case where multiple detections meet the criteria for association with the same landmark, certain detections that would otherwise be associated with a landmark will have to search for a different correspondence (or form a new landmark). To account for this, we use the Hungarian algorithm to solve for the optimal assignments for all detections [33].

4.4.2 Incorporating Visual Input into Data-Association

Applying the Mahalanobis distance metric to solve data-association is effective in the context of building our map, but there are reasons to believe it could be improved. For example, though the vertex detections are made from visual input, we do not take advantage of any visual information when using the Mahalanobis distance for data-association. Imagine a situation where a vertex detection is equidistant (according to the Mahalanobis distance) from two separate vertices in the map. In these cases, often just by looking at the images captured by the panoramic camera, a human would easily be able to correctly match vertex detections where a system relying on other metrics might fail. So, the question becomes, how do we allow our robot to take advantage of the information in an image to improve its solution to the data-association problem?

Fortunately, the problem we are trying to solve — match a feature detected in an image from frame to frame — has been well studied in the field of computer vision. As discussed in Chapter 2.2, feature descriptors map image patches to fixed-dimensional latent spaces. From there, one can measure the distance between descriptors using euclidean distance, and define a threshold for association as one would the Mahalanobis distance. Unfortunately, handcrafted feature descriptors (such as SIFT or ORB) are not well-suited for our task, as the features we are detecting — vertices output from our neural net — are not necessarily well described by these mappings. So, we instead turn to the much newer domain of "learned descriptors."

A learned descriptor, like the one presented in the 2017 paper: "L2-Net" [63], works by passing an image patch through a convolutional neural network, and having the network project that patch into a latent space much like a hand-crafted feature. The difference between learned features and hand-crafted features is that a learned feature uses a convolutional network for its mapping, while traditional features use a human designed mapping. Because the task of matching vertex detections from frame to frame is a unique problem, relying on learned descriptors is a natural approach.

However, it would wasteful to discard information that may be useful in the data-association process and rely solely on visual data. Ideally, we could find a way to incorporate both visual input and a distance in physical space to utilize the advantages of both approaches. In order to do that in a way that does not require hand tuning the influence of different metrics, we propose a network which also learns the mapping from the latent space plus a physical distance to a binary decision of correspondence. In other words, we give our network two images, as well as a physical distance metric, and have it map them to a single probability of association.

4.4.3 Network Architecture

Very recently, approaches in this mold have been developed for learned features. "Match-Net" [23], implements a unified approach that jointly learns a mapping into a fixed-dimensional descriptor space, then uses a learned distance metric to determine the probability of a feature match in and end-to-end approach. Here, we extend that

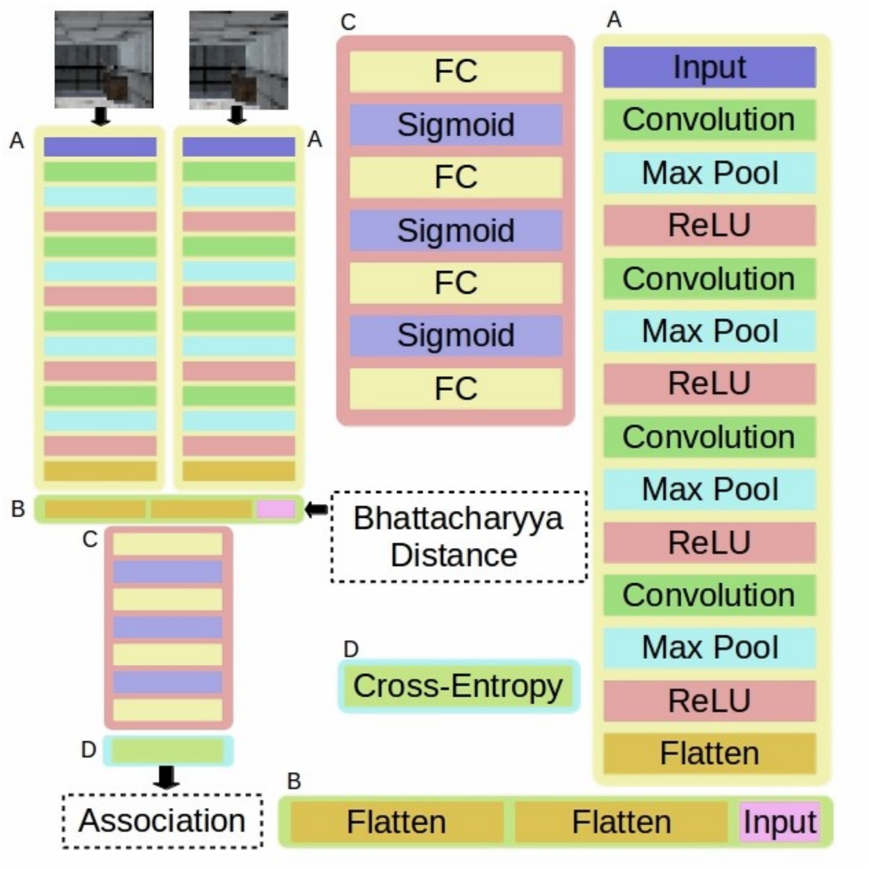

Figure 4-4: Depiction of association network. The full network is depicted on the left, including sample input. The convolutional towers (A), concatenation layer (B),fully connected metric layers (C), and output layer (D) are expanded on the right.

work to, at the latent descriptor layer, add an additional input of the Bhattacharyya distance between vertex detections.

Taking inspiration from the "Match-Net" architecture, our system employs a two-part network composed of a dual-tower convolutional structure with tied parameters (often referred to as a Siamese network) to learn the latent descriptor representation, and a fully connected network to learn the mapping from the latent descriptor layer to a single output measuring the probability of association. Where our networks differ is that, at the transition between the two sections of the network, we introduce an additional parameter to the fully connected network: a physical distance metric between two vertices. In this way, we are capable of incorporating both visual input, and physical distance in data-association. The architecture of our network is depicted in figure 4-4.

The input to each "tower" in our network is based on the output of the learned vertex detector network. For each detected vertex, we take a 28×28 pixel image patch centered around its bearing at the mid-line of the panoramic image. After the robot takes a single step, we repeat the process, and extract the vertex patches for the new panoramic image. We want to know if the vertices these patches depict from each panoramic image are the same. Thus, for each pair of patches across time-steps, we input one to each of the "towers" of the network.

The task of encoding image patches to a fixed-dimensional descriptor requires that the mapping each image patch goes through be the same. In other words, the two convolutional networks in each tower that process the image patches are identical in both their design, and parameters. In the training process, updates to the parameters of either tower would be applied to the shared coefficients in the same way. As a result, if the same image patch were input to each tower, they would each map it to the exact same point in the latent space. This architecture is based on past implementations of "Siamese networks," which also use this design, but stop at the descriptor layer. Each tower is composed of four convolutional layers — of kernel dimension ([3,3,3,32], [3,3,32,64], [3,3,64,64], [3,3,64,128]) – each followed by a max pool layer and a ReLU activation layer. At the end of the final ReLU layer, the output is flattened to a single vector of length 128.

Once the images pass through their respective towers, we continue by concatenating the flattened final layers from each towers along with the normalized Bhattacharyya distance computed between the two vertex detections for each image patch. Bhattacharyya distance is a generalization of the Mahalanobis distance which incorporates the noise from both detections. We use this metric instead of Mahalanobis distance because the network does not have a notion of which patch was detected first. Once these values have been concatenated, this single vector — now of size 257 — is passed through four fully connected layers — 257 to 128 to 32 to 8 — to a single output which represents the probability that the detections represented by the image patches are from the same vertex. At all except the final layer, the output of each fully connected layer is activated by a sigmoid activation function. The final logit also is passed through a sigmoid after the cross-entropy loss is computed by comparing to the ground-truth label.

4.4.4 Training the Network

Generating data for training the association network requires creating "simulated" vertex detections. We cannot simply use the output of the learned vertex detector, because it does not have ground truth labels of association we can use to train (if it did, the data-association problem would be solved). To get around this, we simulate the robot traveling through our environment, extracting the exact locations of vertices relative to the robot from the simulator. We then define a Gaussian around the perfect detection, and draw from it a 2D point which represents the estimated location of our simulated detection. We do the same for the next position of the robot, and because we know from which vertex the noisy detection originated, we have the label needed to train our network.

The robot was simulated in 100,000 different poses scattered through hundreds of different hallway environments. At each pose, the robot takes a 512×128 pixel panoramic image, from which it extracts the perfect vertex detections. For each simulated vertex detection, we extract a 28×28 image patch, as well as the Bhattacharyya distance between it and any detections in the subsequent pose based on a Gaussian defined around it. We collect test data in the same way, being sure to also save the Mahalanobis distance between detections in sequential frames in order to compare the different metrics. As with training, in order to determine the actual rates of success and failure for our different data-association techniques, we must rely on simulated data to compare to the ground-truth labels.

4.4.5 Comparing Data-Association Metrics

Using a data-set of 100,000 poses taken from several different procedurally generated simulated environments with approximately ten vertices visible in each pose (note this was a different set of poses and environments then the training set), we compared our

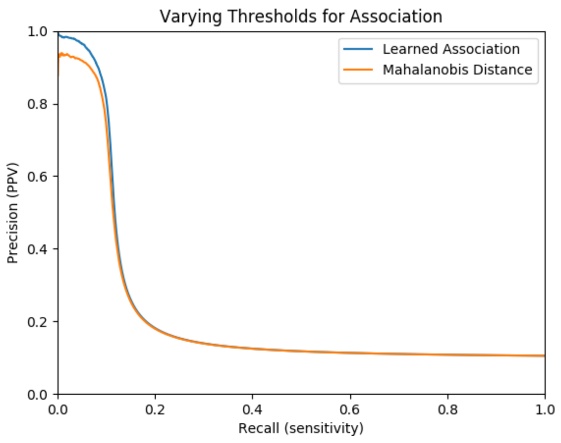

Figure 4-5: The precision recall curve for both the learned association and Mahalanobis distance metric. Each point in the curve represents a different threshold for association. That the learned metric always outperforms Mahalanobis distance regardless of the threshold shows that it is an improvement as a metric for data-association. The area under the curve for the the learned association metric was 0.224, compared to 0.217 for the Mahalanobis distance metric, demonstrating a small, but observable improvement.

learned association network to a baseline of Mahalanobis distance. Because the rates of positive and negative detections depend greatly on how the output of each metric is thresholded, we tested each system on thousands of different thresholds each. The precision-recall curve of both systems is plotted in figure 4-5

At each point on the curve, the learned system outperforms the Mahalanobis distance metric, meaning that at each level of recall, the learned association has a higher level of precision. Though this is not the only method for analyzing the effectiveness of classification metric, it is a useful indicator that we have improved upon our baseline using the learned association network.

However, the above curves also show that both systems are imperfect. To get further insight on where these metrics fail, we can look at the counts of false-positive and false-negative detections. Inspecting the confusion matrices for selected thresholds of 1 for Mahalanobis distance, and 0.5 for the learned metric (Table 4.1), we see that each system exhibits both failures with false-negatives, and false-positives.

Building a map well in the presence of this noise demonstrates the importance of sampling the posterior in our process. Anecdotally, one of the most important aspects of either approach to data-association is the concept of mutual exclusion [42], which

	Mahalanobis Distance		**Learned Association**		
	p	**n**	**p**	**n**	**total**
$\mathbf{p'}$	TP 430	FN 10089	TP 8317	FN 2202	10519
$\mathbf{n'}$	FP 33	TN 89448	FP 1029	TN 88452	89481
total	463	99537	9346	90654	

Table 4.1: Confusion matrices for both the Mahalanobis distance metric thresholded at 1, and the learned association metric thresholded at 0.5. These values are illustrate some of the failure modes of different metrics at different thresholds. Regardless of the metric, the proposed map building procedure is better equipped to handle false-negative detections compared to false-positives.

is valuable in preventing errors when two vertices are very close to each other.

4.5 Discussion

In this chapter, we presented a method for building a sparse map from vision designed explicitly for navigation. This map utilizes a learned action detector in order to identify the high-level actions available to a robot equipped with a panoramic camera, and is capable of fusing those detections into a graph of polygonal vertices and walls in the presence of noise. In order to be robust to noise, a posterior over maps was defined, enabling the robot to sample different possible graphs to find the most likely map given its observations of the world. A major focus of the chapter was the development of a method for data-association in the context of the proposed map. Specifically, a learned association network was presented which was shown to outperform a baseline of Mahalanobis distance for frame-to-frame association.

The analysis of the association metric was based on data collected in simulation. Future work will involve having both the learned action detector and learned association modules working on a robot operating in the real world. Preliminary efforts have found data collection to be particularly challenging however, in part due to the difficulty of getting ground truth labels for the action detector. The ultimate goal will be to integrate the map representation described here with the Learned Subgoal Planner presented in 3. The following chapter (Chapter 5), will discuss how the work presented in this thesis has been related, and consider future directions to integrate and extend the two lines of research.

Chapter 5

Conclusion

This thesis presented work relating to the topic of autonomous navigation of unknown environments. As stated in Chapter 1, this problem can be broken up into two parts. First, the robot must be able to build a representation of the world as it travels, so in the event it has to retrace its steps it can take advantage of past observations to do so efficiently. Second, a robot must be able to reason about paths that enter unknown space in order to make intelligent decisions about how to plan to reach its goal.

Planning in a POMDP framework can be very computationally expensive, however it is a popular way to model the navigation unknown environments because it properly captures the uncertainty of planning through unobserved space. In Chapter 3, the Learned Subgoal Planner was presented to address this problem, and takes advantage of high-level actions for navigation to generate plans which reason about the properties of unknown space. Training on MIT floor-plans, we demonstrated a 21% increase over the naive approach to navigation. Additionally, our technique translates well from simulation to the real world, and was shown to work on an RC car equipped with a laser scanner.

Despite great advancements in the field of robotic mapping, outside of dense representations which can be very computationally taxing, few maps are designed to be useful for the task of navigating partially explored environments. Chapter 4 discussed a novel method for creating a map of an environment from vision, explicitly designed to be used for the task of navigation. A learned action detector was presented, which generates observations of the high-level actions available to a robot. A technique for fusing these detections into a map of actions was presented, and special attention was paid to the process of data-association within that map. Specifically, a learned association metric was proposed, which, inspired by the field of learned feature descriptors, showed improvement over a baseline of the Mahalanobis distance.

One assumption that was made in the map building process of Chapter 4 was that the robot was not tasked with solving the localization problem. In other words, it was presumed to know exactly where it was at all times. Localization was not the focus of this research, but is an important part of map-building, and so must be addressed. One unexplored impact of using visual input for data-association is that it may allow the robot to be robust to drift in estimation as it returns to previously visited locations. If association relied entirely on a physical metric like Mahalanobis distance, errors in state estimation would make the task extremely difficult. The problem of loop closure is something not considered in our work, but by using the learned association network (or some variant thereof), we might hope to allow our map to incorporate loop-closure detections.

Though it was not implemented in this thesis, future work will involve an analysis of the entire map building process, particularly as it relates to the usefulness of the representation to the problem of navigation toward an unseen goal. Though the Learned Subgoal Planner was implemented with an occupancy grid to both represent the world and allow for the extraction of subgoals, in principle any map representation which allows the robot to efficiently navigate known space, while representing high-level actions can be used in the planning framework. Incorporating the action based map with the Learned Subgoal Planner will enable solving navigation tasks in larger scale, which might include obstacles which were outside the range of a laser scanner. Additionally, with a map built from vision, we can also include visual information in the training of the Learned Subgoal Planner. This could allow for more complex tasks, such as finding a specific item in a grocery store, to be solved.

An interesting potential extension to this work would be to include actions which are not explicitly related to navigation in both the Learned Subgoal Planner and the proposed map representation. This would require some method of detecting actions beyond simply rounding a corner, though one could imagine tasks such as opening a door or picking up an object being detectable from a camera image. Including different tasks into a more general map of actions would give a representation which can be used for more complex planning tasks. However, generating data from such actions to train the Learned Subgoal Planner might be more difficult, and a more complex simulator would likely be required to pursue this line of research.

The navigation of unknown environments is a problem which affects many different areas of the field of mobile robotics. This thesis has presented work to address two aspects of that problem — map-building and planning — and showed promising results on both fronts by taking advantage of high-level actions. Continued research with regards to the integration of these works is an important milestone left to be reached in this direction.

Bibliography

[1] *Introduction to Algorithms, Third Edition.* MIT Press, 2009.

[2] I. Arvanitakis, K. Giannousakis, and A. Tzes. Mobile robot navigation in unknown environment based on exploration principles. In *Conference on Control Applications (CCA)*, Sept 2016.

[3] K.J. AstrÃűm. Optimal control of markov decision processes with incomplete state estimation. *J. Math*, pages 174–205.

[4] Abraham Bachrach. *Trajectory Bundle Estimation For Perception-Driven Planning.* PhD thesis, Massachusetts Institute of Technology, 2017.

[5] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[6] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[7] Jose A. Castellanos and Juan D. Tardos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[8] Hsien-Te Cheng. *Algorithms for partially observable Markov decision processes.* PhD thesis, University of British Columbia, 1988.

[9] Wendell H. Chun and Nikolaos Papanikolopoulos. *Robot Surveillance and Security*, pages 1605–1626. Springer International Publishing, Cham, 2016.

[10] R. Craig Conlter. Implementation of the pure pursuit path tracking algorithm, 1992.

[11] Marc Deisenroth and Carl E. Rasmussen. Pilco: A model-based and data efficient approach to policy search. *International Conference on Machine Learning (ICML)*, 2011.

[12] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[13] Arnaud Doucet, Nando de Freitas, Kevin P. Murphy, and Stuart J. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 176–183, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[14] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[15] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989.

[16] Alberto Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.* PhD thesis, Pittsburgh, PA, USA, 1989. AAI9006205.

[17] Hugh F. Durrant-Whyte, Somajyoti Majumder, Sebastian Thrun, Marc De Battista, and Steven Scheding. A bayesian algorithm for simultaneous localisation and map building. pages 49–60, 01 2001.

[18] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[19] D. Fourie, J. Leonard, and M. Kaess. A nonparametric belief solution to the bayes tree. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2189–2196, Oct 2016.

[20] Thierry Fraichard. A short paper about motion safety. *International Conference on Robotics and Automation (ICRA)*, 2007.

[21] M.W.M. Gamini Dissanayake, P M. Newman, Steven Clark, Hugh F. Durrant-Whyte, and M.A. Csorba. A solution to the simultaneous localization and map building (slam) problem. *Robotics and Automation, IEEE Transactions on*, 17:229 – 241, 07 2001.

[22] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[23] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*, pages 3279–3286. IEEE Computer Society, 2015.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[25] Hokuyo. Hokuyo UTM-30LX Scanning Laser Rangefinder. https://www.hokuyo-aut.jp/search/single.php?serial=169, 2018.

[26] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[27] Intel. NUC 7 Enthusiast Mini PC. https://www.intel.com/content/www/us/en/products/boards-kits/nuc/mini-pcs/nuc7i7bnkq.html,.

[28] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[29] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *International Conference in Robotics and Automation (ICRA)*, 2018.

[30] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.

[31] Sertac Karaman and Emilio Frazzoli. High-speed flight in an ergodic forest. *International Conference on Robotics and Automation (ICRA)*, 2012.

[32] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 2013.

[33] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1âĂŘ2):83–97, 1955.

[34] Michael L. Littman. The witness algorithm: Solving partially observable markov decision processes. Technical report, Providence, RI, USA, 1994.

[35] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 362 – 370. 1995.

[36] Jorge Lobo, Lino Marques, Jorge Dias, Urbano Nunes, and Aníbal T. de Almeida. Sensors for mobile robot navigation. In Anibal T. de Almeida and Oussama Khatib, editors, *Autonomous Robotic Systems*, pages 50–81, London, 1998. Springer London.

[37] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[38] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[39] A.A. Markov. Extension of the law of large numbers to quantities, depending on each other (1906). reprint. *Journal ÃĽlectronique d'Histoire des ProbabilitÃľs et de la Statistique [electronic only]*, 2(1b):Article 10, 12 p., electronic only–Article 10, 12 p., electronic only, 2006.

[40] Lord MicroStrain. 3DM-GX4-45âĎć Industrial-Grade All-In-One Navigation Solution. https://www.microstrain.com/inertial/3dm-gx4-45.

[41] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[42] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1985–1991 vol.2, Sep. 2003.

[43] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Eighteenth National Conference on Artificial Intelligence*, pages 593–598, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[44] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61, Jun. 1988.

[45] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *CoRR*, abs/1502.00956, 2015.

[46] Liz Murphy and Paul Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *International Conference on Robotics and Automation (ICRA)*, 2008.

[47] Robin R. Murphy, Satoshi Tadokoro, Daniele Nardi, Adam Jacoff, Paolo Fiorini, Howie Choset, and Aydan M. Erkmen. *Search and Rescue Robotics*, pages 1151–1173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[48] Reem Nasir and Ashraf Elnagar. Gap navigation trees for discovering unknown environments. *Intelligent Control and Automation*, 6(04):229, 2015.

[49] RACECAR/J. RACECAR/J Platform. https://racecarj.com/, 2018.

[50] Ananth Ranganathan and Frank Dellaert. Online probabilistic topological mapping. *The International Journal of Robotics Research*, 30(6):755–771, 2011.

[51] C. Richter, J. Ware, and N. Roy. High-speed autonomous navigation of unknown environments using learned probabilities of collision. In *International Conference on Robotics and Automation (ICRA)*, 2014.

[52] Charles Richter. *Autonomous navigation in unknown environments using machine learning.* PhD thesis, Massachusetts Institute of Technology, 2017.

[53] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. 07 2017.

[54] Nicholas Roy and Caleb Earnest. Dynamic action spaces for information gain maximization in search and exploration. In *American Control Conference (ACC)*, Minneapolis, MN, 2006.

[55] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[56] R. Smith, M. Self, and P. Cheeseman. Autonomous robot vehicles. chapter Estimating Uncertain Spatial Relationships in Robotics, pages 167–193. Springer-Verlag, Berlin, Heidelberg, 1990.

[57] Hauke Strasdat, J Montiel, and Andrew J. Davison. Real-time monocular slam: Why filter? pages 2657–2664, 05 2010.

[58] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[59] Josh D. Tenenberg. *Abstraction in Planning.* PhD thesis, Rochester, NY, USA, 1988. Order No: GAX88-16885.

[60] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics.* MIT press, 2005.

[61] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artif. Intell.*, 99:21–71, 1998.

[62] Sebastian Thrun. Exploring artificial intelligence in the new millennium. chapter Robotic Mapping: A Survey, pages 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[63] Yurun Tian, Bin Fan, and Fuchao Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. pages 6128–6136, 07 2017.

[64] Benjamin Tovar, Luis Guilamo, and Steven M LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Algorithmic Foundations of Robotics VI*, pages 425–440. Springer, 2005.

[65] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[66] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack W. Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Jimenez Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matthew Botvinick, Demis Hassabis, and Timothy P. Lillicrap. Unsupervised predictive memory in a goal-directed agent. *CoRR*, abs/1803.10760, 2018.

[67] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. *J. Artif. Int. Res.*, 58(1):231–266, January 2017.

[68] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.