Policy-Based Planning for Robust Robot Navigation

by

Robert T. Goeddel

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2018

Robert T. Goeddel

rgoeddel@umich.edu

ORCID iD: 0000-0002-4296-9149

DEDICATION

This thesis is dedicated to Tom, Linda, and Kate Goeddel. It was a slog, but your support made it possible.

ACKNOWLEDGMENTS

To my advisor, Ed Olson, and my committee members Chad Jenkins, Matt Johnson-Roberson, and John Laird: thank you for your guidance and wisdom.

To the many members of APRIL I have worked with over the years: I have been lucky to be surrounded by brilliant, funny, and caring people. We've journeyed far, together; from sunny Adelaide, Australia to sweltering Muscatatuck Urban Training Center, Indiana. We've burned the midnight oil to make MAGIC robots map and Smartcarts drive. I can't say I'll miss the late night hacking sessons and paper deadlines, but I'll treasure the icreasingly inane conservations only consecutive days of sleep deprivation can produce.

To the support staff of the Computer Science department: you made the department feel like a family. I always knew you had my back. Thanks for making sure the only thing I had to stress about was my own research.

To my friends during my decade-long stay in Ann Arbor: you are too many to name, but you reminded me that there was a world outside of research. Thank you, all, for making me leave the lab and explore the world around me. There's so much I might never have discovered about Ann Arbor if someone hadn't drug me, kicking and screaming, to check it out. Thank you also for your support during the more stressful parts of grad school. In particular, I'd like to thank Erik Brinkman for always keeping the fridge full of exciting craft beers, Lauren Hinkle for being a sounding board whenever I was full of doubt, and Lizzie Mamantov for always being able to cheer me up, even if she was having a terrible day, herself. You guys kept me sane when I couldn't see the light at the end of the tunnel.

In loving memory of my grandmother, Noni Dalrymple for making this all possible. Your support throughout my undergraduate studies laid the groundwork for all that followed. Thanks, Grandma!

And finally, Mom, Dad, and Kate: thanks for continuing to pick up the phone, even when I was calling several times a week. I know I was a pain, but hey: we made it!

My work during this thesis was supported by ONR Grant N00014-13-1-0217 and by DARPA D13AP00059.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AMN Associative Markov Network

CNN Convolutional Neural Network

CRF Conditional Random Field

DART Direction Approximation through Random Trials

EVG Extended Voronoi Graph

FSM Finite State Machine

GMM Gaussian Mixture Model

IRL Inverse Reinforcement Learning

iSAM Incremental Smoothing and Mapping

LfD Learning from Demonstration

LWR Locally Weighted Regression

MPDM Multi-Policy Decision Making

POMDP Partially-Observation Markov Decision Process

SLAM Simultaneous Localization and Mapping

SSH Spatial Semantic Hierarchy

SVM Support Vector Machine

VRF Voronoi Random Field

RL Reinforcement Learning

RRT Rapidly-exporing Random Tree

SGD Stochastic Gradient Descent

ABSTRACT

This thesis proposes techniques for constructing and implementing an extensible navigation framework suitable for operating alongside or in place of traditional navigation systems. Robot navigation is only possible when many subsystems work in tandem such as localization and mapping, motion planning, control, and object tracking. Errors in any one of these subsystems can result in the robot failing to accomplish its task, oftentimes requiring human interventions that diminish the benefits theoretically provided by autonomous robotic systems.

Our first contribution is Direction Approximation through Random Trials (DART), a method for generating human-followable navigation instructions optimized for followability instead of traditional metrics such as path length. We show how this strategy can be extended to robot navigation planning, allowing the robot to compute the sequence of control policies and switching conditions maximizing the likelihood with which the robot will reach its goal. This technique allows robots to select plans based on reliability in addition to efficiency, avoiding error-prone actions or areas of the environment. We also show how DART can be used to build compact, topological maps of its environments, offering opportunities to scale to larger environments.

DART depends on the existence of a set of behaviors and switching conditions describing ways the robot can move through an environment. In the remainder of this thesis, we present methods for learning these behaviors and conditions in indoor environments. To support landmark-based navigation, we show how to train a Convolutional Neural Network (CNN) to distinguish between semantically labeled 2D occupancy grids generated from LIDAR data. By providing the robot the ability to recognize specific classes of places based on human labels, not only do we support transitioning between control laws, but also provide hooks for human-aided instruction and direction.

Additionally, we suggest a subset of behaviors that provide DART with a sufficient set of actions to navigate in most indoor environments and introduce a method to learn these behaviors from teleloperated demonstrations. Our method learns a cost function suitable for integration into gradient-based control schemes. This enables the robot to execute behaviors in the absence of global knowledge. We present results demonstrating these behaviors working in several environments with varied structure, indicating that they generalize well to new environments.

This work was motivated by the weaknesses and brittleness of many state-of-the-art navigation systems. Reliable navigation is the foundation of any mobile robotic system. It provides access to larger work spaces and enables a wide variety of tasks. Even though navigation systems have continued to improve, catastrophic failures can still occur (e.g. due to an incorrect loop closure) that limit their reliability. Furthermore, as work areas approach the scale of kilometers, constructing and operating on precise localization maps becomes expensive. These limitations prevent large scale deployments of robots outside of controlled settings and laboratory environments.

The work presented in this thesis is intended to augment or replace traditional navigation systems to mitigate concerns about scalability and reliability by considering the effects of navigation failures for particular actions. By considering these effects when evaluating the actions to take, our framework can adapt navigation strategies to best take advantage of the capabilities of the robot in a given environment. A natural output of our framework is a topological network of actions and switching conditions, providing compact representations of work areas suitable for fast, scalable planning.

CHAPTER 1

Introduction

A typical robot system is dependent upon the reliable operation of its localization, planning, and control systems. While these systems have all improved dramatically over time, long-term robustness remains an issue. Many improvements target performance in the common case, but add new catastrophic error modes: Simultaneous Localization and Mapping (SLAM) methods can greatly reduce positioning error, but can fail catastrophically when an incorrect loop closure occurs; control systems with detailed models can achieve high performance, but can perform poorly when the model parameters are no longer accurate. A central idea is that there are often multiple ways for a robot to complete its task, and that each of these methods are likely to have different likelihoods of failure. Sometimes, a simpler method (that lacks catastrophic failure modes) can yield sufficiently good performance, and thus is be a better choice than a high performance method. Other times, failure modes are dependent upon the environment, and intelligently switching between those methods can reduce the likelihood of failure.

This thesis describes an integrated planning and control framework. The key idea is to develop a suite of behaviors that a planner can stitch together to solve a particular problem. Critically, the planner makes predictions about the performance of these behaviors based on the particular task and the current state. An important effect of this approach is that the robot can dynamically switch between relative simple (but robust) behaviors and more capable (but fragile) methods, using the latter only when necessary. An effect of this approach is that we can often compute solutions to tasks using *only* simple methods; since these are relatively easy to implement, our planning framework has the effect of making it easier to build highly-capable robots.

Creating an extensible system introduces many challenges not faced by the designers of specialized systems, particularly in the domains of sensing and actuation. An industrial welding robot, for example, need only enough information to perform the same set of welds repeatedly. It does not require a detailed map of its surroundings, and may even be able to operate without any active obstacle avoidance systems. Conversely, robots deployed in more general settings must understand their surroundings to be able to act effectively in the environment. While advances in recognition and mapping have greatly improved upon and extended the range of capabilities available to fully-autonomous platforms [76, 36], these methods are not error free. Misclassification of objects or poor data association can lead to erroneous actions or robots becoming lost. While errors cannot be suppressed entirely, some can be predicted, offering opportunities to adapt robot behavior to mitigate their severity.

Another limitation of existing systems is that most require expert users to program or teach new tasks. Expertise in the necessary areas is in limited supply, presenting challenges to widespread deployment of robotic systems. If general-purpose robotic systems are to gain traction, there must be a paradigm shift towards methods supporting on-the-fly learning. While some of this can be accomplished through unsupervised learning and exploration within the robot's operational domain, in many cases, direct human interaction presents a more expedient avenue for acquiring knowledge. The typical user will be a non-expert, so an accessible interface such as language is desirable. The benefits of this interface are twofold: it supports the learning of new tasks as well as provides a medium for engaging a human when assistance is required.

In this thesis, we focus our efforts on providing an extensible framework to support navigation-based tasks. A broad variety of robotic domains are predominantly navigation based, such as providing tours, making deliveries, and mapping or surveying. Our specific contributions are in the areas of reliability and scalability. Reliability of systems is critical to ensure that they can operate safely for long durations of time. Scalable representations of the world allow robotic systems to operate in a wider variety of environments. Though not the primary focus of the design, our system is also structured with natural language interfaces in mind, offering future opportunities to explore the problem of accessibility to non-experts.

We open with a brief introduction to robot motion planning and navigation in Chapter 2. This chapter describes a standard navigation pipeline and how it relates to the contributions of this thesis. In Chapters 3 and 4, we introduce our planning framework, which we call DART. DART constructs its plans based on the set of closed-loop control policies (e.g. follow the left wall) and switching conditions (e.g. until you reach an intersection) available to the robot. This set of capabilities can be adapted or extended to suit the needs of the robot in a particular environment. While a fixed set of pre-constructed skills may be suitable for many applications, the ability to learn new skills offers the opportunity for a robot to adapt to new challenges. In the latter portion of the thesis, we introduce a pair of methods suitable for learning these core capabilities beginning with Chapter 5, which introduces a technique for learning a semantic place classifier intended for use as a switching condition. Then, in Chapter 6, we describe a method for learning control policies from user demonstrations. In the final chapter, we summarize the contributions and discuss how the work can be extended in the future.

1.1 Planning for Imperfect Execution (Ch. 3 and Ch. 4)

The reliability of a robotic system is tied to its ability to sense and actuate in the environment. Electro-mechanical advances to both sensors and robotic platforms themselves offer one potential avenue for improvement. For example, over the past two decades, robots have moved from predominantly sonar-based sensors [52, 99] to LIDAR [102, 77, 33], which provide more precise measurements of structure around the robot. Dense RGB-D sensors like the Kinect have also proven popular [39, 56, 87], both in object classification and mapping tasks.

Despite continual improvement to robot sensing capabilities, sensor noise and errors will always be present. Sensing imperfections percolate throughout the navigation pipeline, inducing errors in systems consuming the sensor data. These systems may themselves be imperfect, compounding the issue. Sometimes the resulting errors are harmless, but in the worst cases, they may result in critical misunderstandings of the environment, such as misclassifying a stop sign as a speed limit sign [22]. Robot actuation is also imperfect, adding yet another source of error. If the resulting errors are sufficiently severe, human intervention may be necessary to help the robot recover and resume normal operation. These interventions detract from the value of an autonomous system by taking away time from people who often have their own tasks to complete. System designs try to avoid interventions by driving down error rates.

An alternative to constructing perfect hardware is to adapt algorithms to mitigate problems arising from sensing and actuation errors. Chapter 3 describes the strategy employed in this thesis: to model and simulate possible errors in an attempt to predict their effects. Based on these predictions, the robot can compensate for or entirely avoid potential failures. We introduce an algorithm called DART which generates route descriptions based on the reliability with which it believes they can be followed. DART employs Monte Carlo simulations and a black-box model of the wayfinder (the agent following the directions) to predict errors in execution the wayfinder might make while following a set of directions. Our key result shows that the algorithm is able to adapt the route and its description to take advantage of the strengths of the modeled wayfinder and mitigate issues caused by the wayfinder's weaknesses. It is straightforward to extend the model to support arbitrary sets of directions, allowing the algorithm to be applied to a variety of environments.

In Chapter 4, we describe a strategy for implementing the ideas from Chapter 3 in a robotic domain. The key insight is that the directions being optimized in DART can be recast as a chain of control policies and switching conditions. We demonstrate this system working in simulated environments and discuss how the resulting plans can be encoded as a topological map for future use. DART-style planning is most efficiently accomplished with topological map representations storing the types and results of possible actions that can be taken from a variety of key locations. Topological maps offer a sparse encoding of the environment compared to metric maps but can be

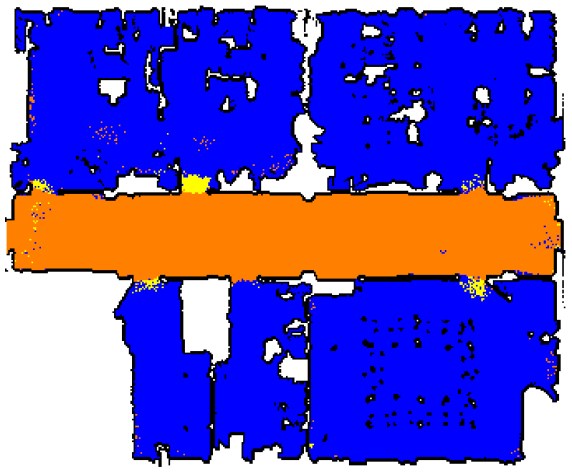


Figure 1.1: Learned semantic place labeling. Different colors indicate different classes of places, in this case “room”, “corridor”, and “doorway.” Semantically labeled places can be used as landmarks, allowing the robot to detect when it has reached a critical decision point in its route.

challenging to construct.

Our method presupposes the existence of easily-modeled control policies and switching conditions. To support the needs of DART, we must be able to accurately simulate the execution of sequences of policies and switching conditions. In Chapters 3 and 4, we employed solutions such as carefully hand-coded behaviors and environments augmented by visual fiducials to meet our needs. While this strategy may prove sufficient in some cases, it does not scale well to supporting a wide variety of environments. In the following chapters, we discuss strategies for using learned behaviors and switching conditions.

1.2 Learning Switching Conditions (Ch. 5)

A switching condition provides a signal to the robot that it time to change control policies. The types of switching conditions provided to DART greatly influence the structure of the routes selected and the reliability of the resulting system. Recognizing key places (landmarks) or the transitions between them can provide valuable cues about when to switch policies. This type of cue is commonly used in high-quality directions (go until you reach a T-intersection), as they are often easy to detect and provide useful localization information. The contribution of Chapter 5 is a method for learning a semantic place classifier suitable for use in detecting this style of switching condition in the DART framework.

Though it is not always critical for a robot to possess a semantic understanding of its surroundings, such knowledge can be convenient in constructing human-friendly interfaces. A shared knowledge of classes of landmark such as “corridor” or “doorway” enables bi-directional route

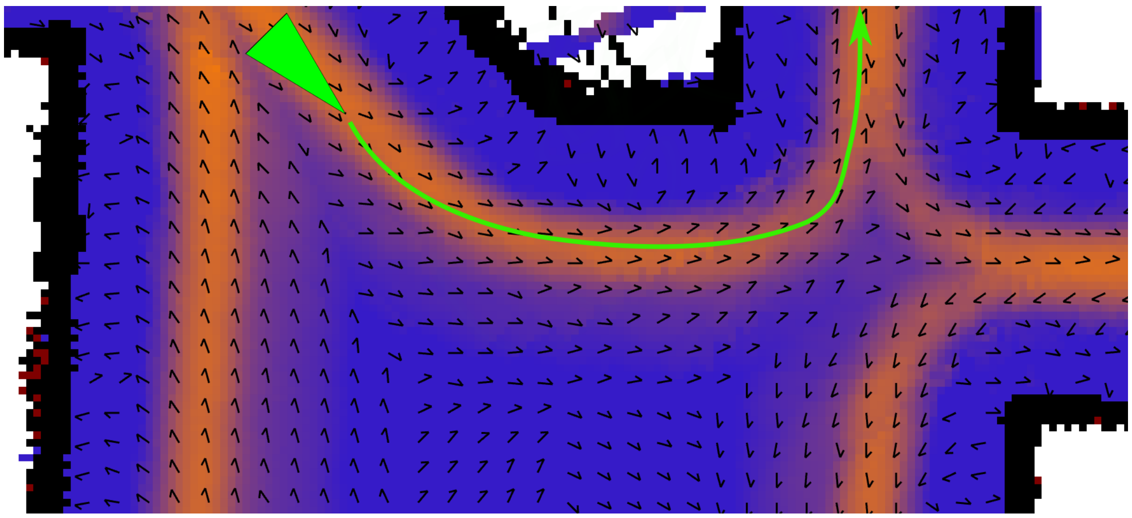


Figure 1.2: A learned reward function for a demonstrated behavior. By moving to high-reward states, the robot (green) can reproduce the behavior (left wall following). Orange denotes high reward locations, while blue denotes low reward. The black vector field indicates the locally optimal orientation for achieving a high reward in at a particular coordinate.

communication between humans and robots operating in the same environment. While we do not take advantage of such semantic information in this thesis, we train our network based on a subjective semantic labeling of the environment to support the development of future linguistic interfaces.

We present results for our classifier based on a dataset introduced in [71], limited to the provided semantic classes "room", "corridor," and "doorway." Our method performs comparably to prior methods, but without reliance on hand-engineering features. The contributed landmark detector is well-suited for integration within the DART framework, as its sensing needs are straightforward to support in simulation and is computationally inexpensive.

1.3 Learning Control Policies (Ch. 6)

In addition to needing to know when key locations have been reached, DART requires that the robot have a set of control policies for moving through the environment. The control policies made available to the robot dictate where it may move in the environment and how it gets there. Hand-coded policies may not perform well in all environments, so it is desirable that control policies can be learned that generalize well to new environments, or can be extended when they do not.

In Chapter 6, we propose a pair of navigation methods suitable for navigation in most indoor environments and introduce a technique for learning these abilities from teleoperated demonstrations. Demonstrations are an appealing method for learning in this application, as they can often be provided by experts and non-experts alike, offering opportunities for robots to learn behaviors once deployed in their environments.

We contribute a simple but effective way of learning a control policy in the form of “behavior functions” suitable for integration in a potential-field navigation framework. We compose the learned behavior functions with a traditional repulsive function to make safe behaviors capable of producing trajectories similar to those from the training demonstrations. To show that the policies generalize well, we demonstrate them in several environments including one never before seen by the robot. As with the semantic place classifier, the resulting control policies are well-suited for use in DART, as their input data is straightforward to synthesize and control updates are efficient to compute.

CHAPTER 2

Background

A robot's position and orientation in the environment is its *pose*. The objective of a navigation system is to guide a robot from its current pose to (within some threshold of) some target pose. To accomplish this goal, the robot must determine a traversable route between its current and target pose and then drive along it, avoiding any unanticipated hazards along the way.

This thesis is about building a robust and scalable robotic navigation system. Navigation is a challenging task, requiring many individual robotic capabilities to come together to enable even the most basic functionality. We make contributions at several layers of this stack of sub-systems. In the following chapter, we describe a typical organization for a robotic navigation pipeline and introduce key terms and concepts for understanding its sub-components.

In most cases, the target pose is beyond the robot's current sensor horizon. Typically, navigation systems employ a map to support planning outside the robot's limited field of view, requiring the robot to be *localized* within that map, i.e. to know how its current pose is related to the frame of the map. This necessitates the existence of some form of map describing the structure of the environment. Robotic systems primarily employ two types of map representations: detailed metric maps encoding the precise location of structure in the environment and topological maps encoding the relationships between key places.

Once properly localized, the robot must search for a plan which will guide it to the target pose. A plan is typically composed of a sequence of *states* (frequently poses) or *actions* necessary to move between states. Plans are computed to minimize some cost metric, often the time the robot must travel to reach the goal.

Finally, the robot must put its plan into action. In one common representation, the plan consists of a sequence of poses the robot must visit to reach its goal. To execute this plan, the robot employs a controller to issues motor commands guiding the robot along the specified trajectory.

2.1 Mapping the Environment

Maps provide the robot with knowledge about the traversability of the environment. In this thesis, we limit our scope to two-dimensional mapping and navigation tasks, but most concepts can be extended to three-dimensional problems.

The choice of map representation plays a critical role in design decisions for the rest of the navigation pipeline. Maps come in two main flavors: *metric* and *topological*. Metric maps (such as the one seen in Fig. 2.1) construct detailed spatial representations of the world, frequently quantizing the world into grid cells classifying the type of space they represent. Common spatial classifications are "unknown", "free space", and "occupied". The challenge of building precise metric maps is often referred to as the SLAM problem.

Topological maps encode different *places* in the environment and the *transitions* between them [53]. These encodings may include some metric information (e.g. the distance between two nodes), but some encodings do not. For example, the topological map in Fig. 2.2 encodes metric locations on a road network and the valid transitions between them, defining a series of routes that follow the lanes of the roads.

There are a variety of strategies employed by SLAM systems to support robust, large-scale mapping. For example, FastSLAM introduces a tree-based structure to allow it to scale computation time logarithmically in the number of observed landmarks, rather than linearly as seen previous solutions [69]. Incremental Smoothing and Mapping (iSAM) and iSAM 2 are designed to solve SLAM problems efficiently in large-scale, online applications [44, 43]. The key insight of these methods is that oftentimes, unnecessary operations can be avoided by performing incremental updates to a previously factorized matrix.

In addition to methods that allow SLAM to scale well to large problems, many strategies have been introduced to make mapping systems more robust to erroneous loop closures, which often catastrophically distort mapping results. For example, Olson and Agarwal introduce a method for incorporating mixture models into SLAM solutions [76]. Mixtures allow for richer representations of observation models, including the possibility of erroneous observations. This allows the back-end optimization system to robustly handle bad loop closures naturally as part of solving for the maximum likelihood solution for the map. Another strategy called "switchable constraints" is presented by Sunderhauf and Protzel [96]. Rather than modifying the observation models, the authors introduce a new parameter to be optimized by the SLAM back-end allowing it to turn specific loop closures "on" or "off". In this manner, they can discard outliers from their map solutions.

In this thesis, we assume access to a state-of-the-art SLAM system, but do not rely on a particular implementation. In particular, we rely on detailed metrical maps of buildings produced by

SLAM to support simulations for evaluating actions to take in those environments.

Metric maps are useful when extreme precision is necessary, but come with disadvantages. Noisy or erroneous measurements can lead to inconsistencies in the map. Increasing the resolution of the map provides more detail for map consumers, but also leads increased storage and computational needs. Computation and storage costs also increase rapidly in proportion to the scale of the environment being represented, introducing practical limitations to the size of the domain easily represented with a single map. More critically, as the scale of the environment being mapped grows, so does the chance of incorporating an irrecoverably bad loop closure, potentially corrupting the map beyond use. These challenges have lead many to suggest using topological maps in many navigation tasks [53, 100, 89].

Topological maps offer several advantages over their metric counterparts. They can be stored as sparse, graph-based representations of the environment, allowing them to scale up to much larger operational domains. They are also cheap to plan through, since a single transition between places may correspond to a large transition in metric space. The main challenges lie in automatically constructing useful topological maps, as the space of possible topologies given a set of observations is often intractably large. To address this concern, Ranganathan and Delaert employ a sampling-based strategy similar to a particle filter to estimate the distribution of hypothetical topologies [83]. Their method is efficient and frequently finds the correct solution, but due to its reliance on randomly sampled particles, cannot be guaranteed to sample the correct solution. Alternatively, Johnson and Kuipers introduce a heuristic strategy for searching through a tree of hypothesized topologies [42]. Their proposed heuristics allow them to expand and evaluate the tree efficiently, focusing updates on the most probable hypotheses without discarding potentially correct ones.

In this thesis, we use a mixture of both metric and topological maps. We produce building-scale metric maps, which are subsequently used to support simulations used to construct topological maps for use in global planning. These topological maps can later be stitched together to support larger-scale planning.

2.2 Localization

Localization is the process by which a robot determines its position within a map. Localization typically relies on landmarks or unique structure in the environment to disambiguate the robot's current location. One popular localization strategy is to use particle filters [26], which use Monte Carlo sampling techniques to construct and track a distribution of states the robot is currently likely to be in. Other methods rely on scan matching techniques estimating the robot's current pose by trying to directly compute the alignment the robot's current observations with previous

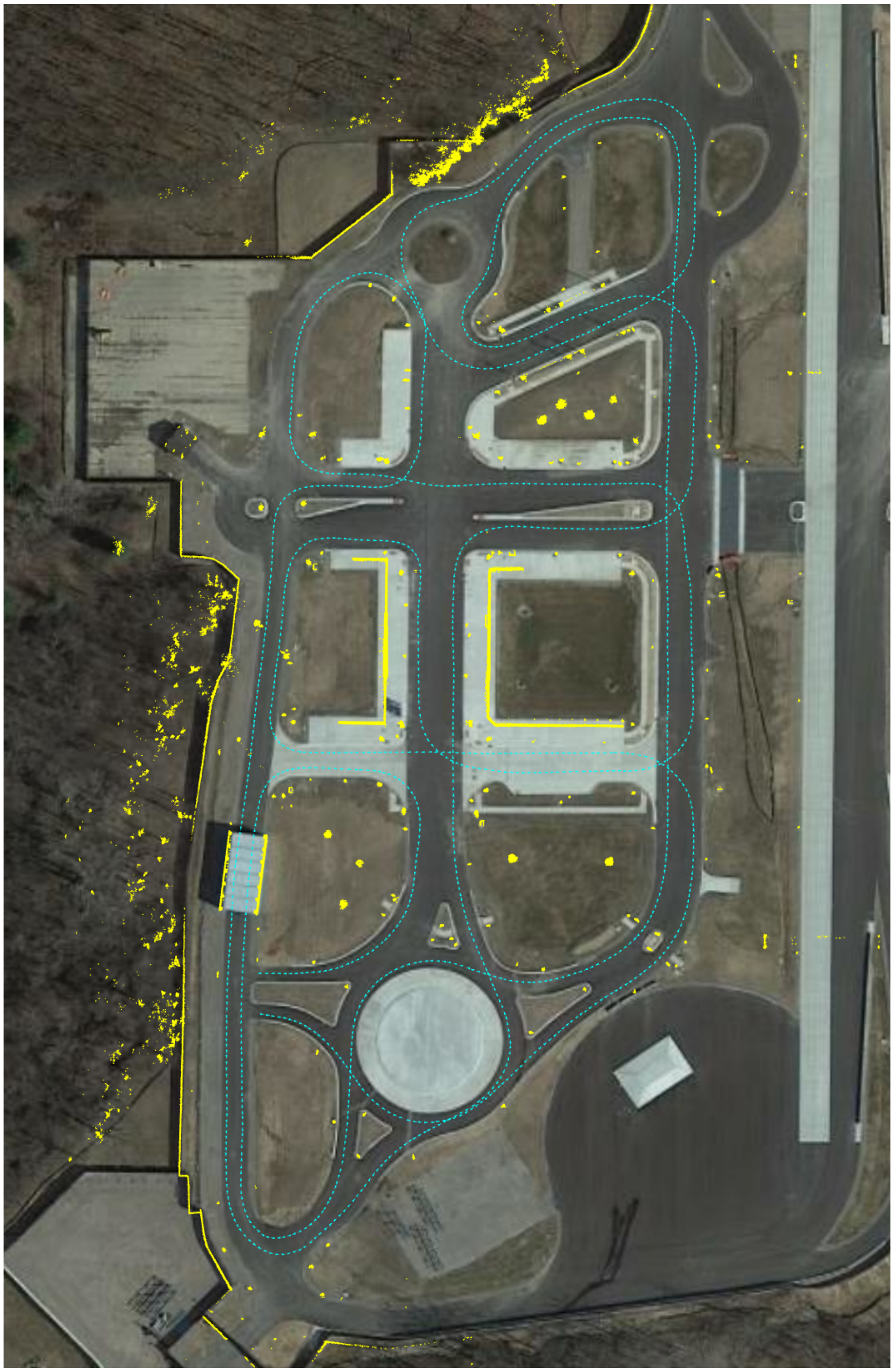


Figure 2.1: A satellite map of MCity overlaid with a 2D metric map of the structure in the environment. The map was built from data collected by driving the route pictured in the environment (cyan) and using LIDAR to detect vertical structure in the environment. An offline SLAM system was then used to construct the maximum likelihood map based on these observations.

viewpoints [75]. One family of these techniques called *Iterative Closest Point* iteratively optimizes over correspondences between the raw points in a pair of scans [9, 87]. Others first classify likely useful structure in the data, discarding irrelevant points [105, 30].

Systems localizing against metric maps are typically designed to estimate the pose of the robot in map coordinates as accurately as possible. Depending on the needs of a system, it might be important to estimate the pose within centimeters of truth. In environments with distinct structure, accurate localization is usually straightforward. However certain layouts of structure (e.g. long, featureless hallways) can be challenging, as viewpoints from different poses along the hallway are similar. Localization errors can lead to task failures, as many navigation systems rely on knowing precisely where the robot is at all times.

Conversely, a topological localization system need only provide high-level information about the current place the robot is in or transition it is making. A key insight is that accurate localization is often not necessary to navigate between places of interest; instead, the robot need only recognize these critical locations. This has the advantage of eliminating the need for precise metric localization, but places increased importance on landmark/place selection and detection.

In this thesis, we assume that robots typically begin well localized within a metric map of the environment, but make no assumptions about their ability to maintain this level of localization in the future. We provide tools for building a complementary topological representation of the metrically mapped environment and suggest a possible place classifier which could be used to support localization in a topological map.

2.3 Motion Planning

Motion planning is the problem of determining a set of movements which will allow a robot to complete some task. Sophisticated motion planners take into account various environmental and dynamic constraints imposed upon a robot to compute achievable plans [58]. In this thesis, we focus on the particular task of moving a mobile robot through its environment, but motion planners can also be used in tasks such as the grasping or throwing of objects. An example of a motion plan for navigating a road network can be seen in Fig. 2.2.

The object of the motion planner is to compute a collision-free path through the environment. In metric maps, this can be done by searching for connected routes through the *configuration space*, the space of all legal states. In many cases, a simple search strategy such as A^* will perform well [35], but for problems with large, complex search spaces, sampling based methods such as Rapidly-exporing Random Tree (RRT) or Probabilistic Roadmaps may be employed [59, 45]. Another space-reduction technique is to plan in the scale of larger-scale actions. For example, macro-actions have proven popular for making Partially-Observation Markov Deci-

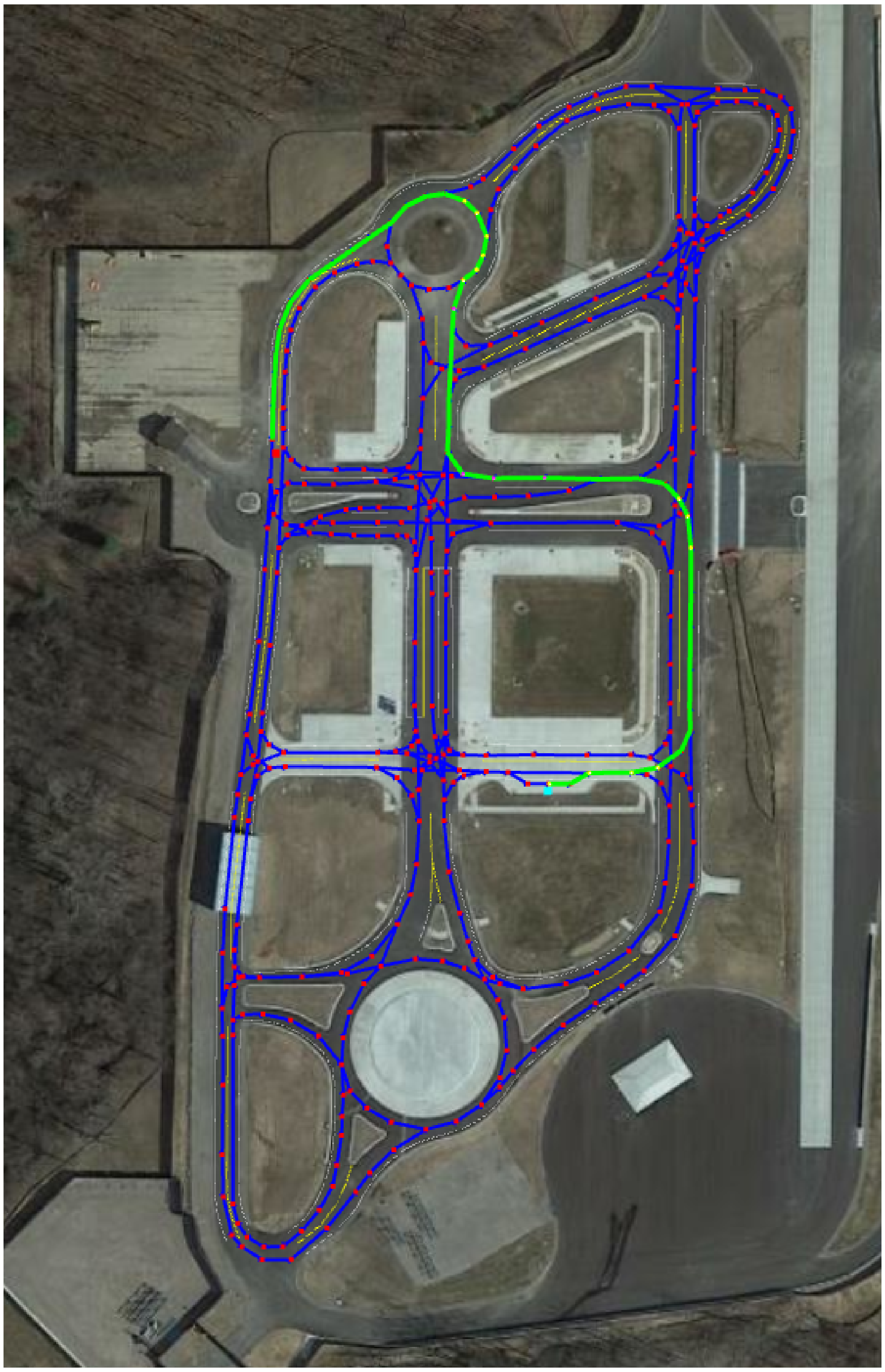


Figure 2.2: A satellite map of MCity overlaid with a topological map of the road network. The topological map encodes a number of drivable places (red dots) and valid transitions between them (blue lines). A route planned through the map is pictured in green. Though many topological maps only encode the relationships between places, in this case, places also encode a coordinate in the world used by downstream path-following systems.

sion Process (POMDP)-style planning tractable [37]. For many-DOF robots, such as manipulators and legged platforms, dynamic motion primitives are often used as building blocks which can be composed into more complex motion plans [15].

Fixed motion plans rely on the environment remaining static during execution, an impossible guarantee in many settings. Several common strategies are employed to account for dynamic changes to the environment. The most straightforward strategy is to re-plan periodically. This allows the robot to adjust to new information collected during execution, but is computationally wasteful, since large portions of the plan are often unchanged between re-plans. A more efficient strategy is to repair the plan, updating only portions of the plan which are obstructed [61, 23].

In topological planning scenarios, the map often includes information about how to transition from one state to another. For example, the Spatial Semantic Hierarchy (SSH) identifies locations in the environment in which the robot enters a stable limit cycle when following certain control policies [52]. A topology of can be constructed from these key locations, with the control policies forming the state transitions. Graph-search algorithms can be used to extract valid routes through the topology. This is the strategy employed in this thesis. Our maps encode macro-actions describing a policy which can guide a robot across large distances to another state. Topologically computed plans are usually compact compared to plans computed in equivalent metric maps, which may encode routes at with step resolutions approaching 5 cm. This makes them well-suited for planning routes through large environments.

2.4 Robot Control

Depending on the level of detail provided by a motion plan, the robot may employ different strategies for executing the desired actions. Ideally, the robot can execute a motion plan exactly, but in reality, imperfect sensing and actuation lead to deviation from the specified actions. Most robots employ one or more controllers to minimize deviation between the motion plan and execution [92]. While some motions are planned directly in actuation space (e.g. motor or joint commands), it is also common for plans to consist of sequences of poses the robot must visit. In these instances, the controller must also generate motor commands based on a model of robot motion to move the robot along the specified path. Another strategy is to compose macro-actions (such as "follow the sidewalk") into motion plans, in which case the robot must have a method for selecting the trajectory or motor commands which best allow it to pursue the desired policy.

An alternative formulation is to skip global motion planning in favor of employing greedy control strategies. Frequently, these are implemented in the form of potential functions used to encode the cost of visiting particular portions of the environment [46, 7, 29]. Attractive potential fields reward the robot for moving towards its goal, while repulsive functions penalize motion that brings the robot close to hazards. Composing these functions results in a cost surface sloping towards the goal and away from obstacles. By computing the gradient on the slope at its current location, the robot can determine the optimal direction to travel and generate appropriate motor commands.

Potential field methods have the advantage of being inherently reactive to changes in the environment, as they only consider the robot's immediate observations when computing the next action to take. This comes at the cost of global reliability: the robot can easily become caught in local minima and fail to react its goal [49]. As a result, these methods have often been put aside in favor of global planning algorithms with proof of *completeness*; a guarantee to find a path if it exists.

In this thesis, we propose employing potential-based controllers to produce closed-loop behaviors such hall following or wall following. Though these behaviors lack global completeness, they are robust to perturbation and can be executed in the absence of global knowledge. This makes them appealing building blocks for DART, which can stitch together sequences of control actions to produce more complex, global navigation behaviors. These plans can be executed with high reliability because they are constructed with full knowledge of the shortcomings of the building blocks.

CHAPTER 3

Generating Reliable Directions [1]

3.1 Introduction

Reliable and efficient navigation abilities are a fundamental building block in mobile robotic systems. They expand the work space of the robot and ensure that it reaches its destination safely. It is standard practice for robotic systems to plan routes optimizing for efficiency, but less common for them to also consider the likelihood the plan will be successfully executed. In this chapter, we re-cast the problem of planning robust plans for navigation as the challenge of giving someone the best set of directions.

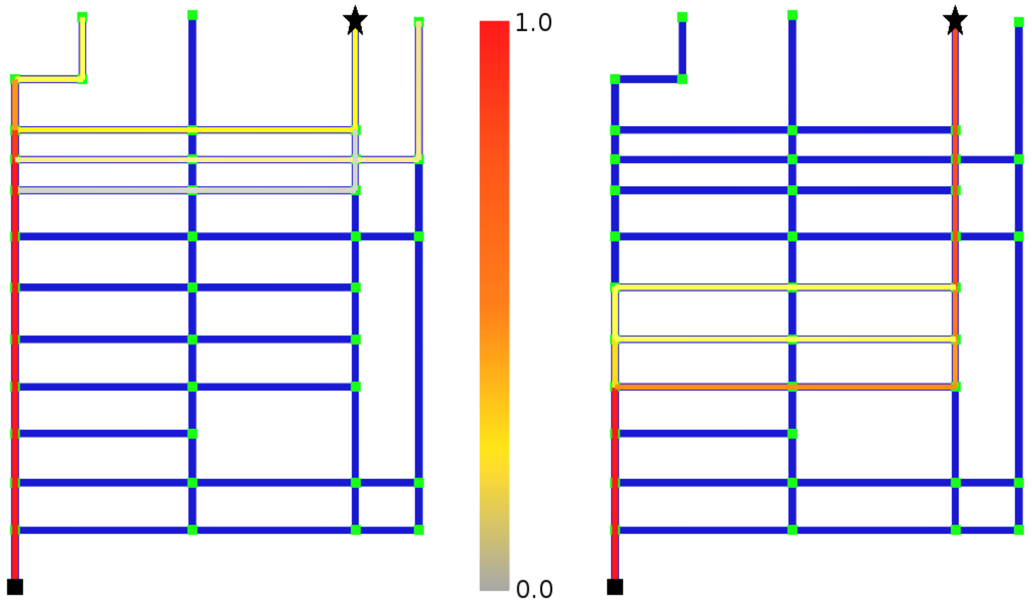
In the context of robotics, a direction can be thought of as a composition of a control policy (follow the road) and one or more switching conditions (until you reach the third light). Constructing the optimal set of directions is therefore analogous to computing the optimal chain of policies and switching conditions.

Effectively directing someone to a particular destination is a difficult task requiring clear and concise directions. Predicting what directions will be most useful is challenging, but generative models offer a powerful tool towards accomplishing this goal. By using a generative model to describe the potential actions made by the *wayfinder* (the person following the directions), we can compute plans based on predictions about those actions.

Modern GPS navigation systems on phones and deployed in cars typically optimize over travel time or distance traveled instead of focusing on the ease of following the directions. Verbal turn reminders and visual depictions of the actions in question mitigate the difficulties in following these routes that may arise from sub-optimal routing through confusing areas. Easy-to-follow plans should be robust to lacking these cues and can be applied to GPS denied situations or to other un-instrumented environments.

Take, for example, a regional hospital. These hospitals are often large and full of difficult-

[1] © 2012. Adapted from Robert Goeddel and Edwin Olson, "DART: A Particle-based Method for Generating Easy-to-Follow Directions," October 2012.



(a) Bad shortest path directions (b) Key (c) Best shortest path directions

Figure 3.1: Simulated wayfinder routes for two sets of shortest-path directions. Paths taken by simulated wayfinders for two sets of shortest-path directions are shown, with colors shown in (b) denoting the percentages of simulated wayfinders traversing a given path segment. The maximum likelihood set of directions may still result in many wayfinders deviating from the specified route. By exploiting knowledge of the wayfinder model to select error tolerant directions, (c) guides more wayfinders to the goal than (a).

to-navigate hallways. Hospitals typically have staffers handle queries and direct people to their destinations. However, this could be done efficiently and effectively by guide robots stationed at the entrances, as well. Such a system could return information about the locations of patients, particular wards, or the offices of certain doctors. Indeed, the only missing piece is the ability to generate a clear set of directions about how to get to the location of interest.

In this chapter, we will motivate and formulate a method for incorporating generative models of wayfinders into the direction-planning process. We will discuss scalability issues and propose optimization techniques for tackling these problems. Finally, we will present a method we call Direction Approximation through Random Trials, or DART, that uses particle-based estimation techniques to find effective sets of directions between points in a map. Action costs are determined by a generative model describing the wayfinder and the environment, rather than a fixed action cost DART approximates the search for the maximum likelihood solution to the problem, rapidly providing paths that take advantage of landmarks and other environmental structure to guide the modeled wayfinder to their goal. We take the wayfinder capabilities as input to our system and consider the problem of finding the best directions given those capabilities. As a consequence, our method builds upon work done by MacMahon [64] and Walter et al. [109]. Our main contributions are:

- The formulation of the wayfinding problem incorporating a generative probabilistic model capable of approximating human wayfinding capabilities,
- Strategies for optimizing depth-first search in the context of wayfinding,
- DART, a tractable, particle-based method for computing directions optimized based on a supplied generative model of the wayfinder and the environment, and
- Evaluation of DART's performance and a framework for evaluating the effectiveness of a set of directions.

3.2 Related Work

3.2.1 Understanding Directions in the Context of Environment

Extensive effort has gone into studying the process of constructing "good" directions. Lynch contributed early thoughts to the literature in his book, *The Image of the City* [63]. Lynch observed that knowledge of environmental structure plays an important role in wayfinding.

Kuipers explored the cognitive maps of humans and how we navigate through the world [51]. With TOUR, he modeled human understanding of their surrounding environment as well as the process with which they navigate through that environment. Others have explored internal world representations and their impact on giving and following directions, finding that, even given human weaknesses in preserving spatial information, efficient and effective direction generation is closely linked with strong spatial abilities [106, 62].

MacMahon created MARCO, an agent designed to follow natural language directions [64]. MARCO models human reasoning about verbal directions, which are often unclear or incomplete. Particularly for directions considered "high quality" by human evaluators, MARCO was able to follow these directions with near-human consistency. Models such as MARCO enable accurate evaluation of direction quality.

3.2.2 Wayfinder Abilities

People have been shown to process different types of directions with varying levels of quality. Though humans have a tendency to give street directions in terms of "go N blocks and turn," these directions are difficult for people to follow correctly [95]. Ambiguity about what is meant by a block and inherent difficulties in recalling specific numbers often lead wayfinders to make errors when following this style of direction. It has further been shown by Hund and Minarik that wayfinding abilities vary noticeably with gender, with men typically navigating more quickly than women [40].

Metrically based directions are popular in GPS navigation systems and in other commonly used routing tools, but humans have been shown to be inaccurate estimators of distance. Cohen et al. showed that geography plays a role in distance estimation [16]. Thorndyke showed that distance estimation error could be tied to the amount of clutter in the environment [98].

Landmarks have been shown to be useful as navigational aids. Directions often use landmarks to guide followers through tricky regions or to help localize them, and extensive work has been done to determine when and why such landmarks are useful [17, 68, 62]. Further consideration has been given to identifying landmarks and automatically extracting them from the environment [82, 12].

Despite this knowledge, automatically generated directions frequently rely on abilities known to be challenging for human wayfinders. While in some scenarios, these directions may still be effective, direction quality could be improved by using more suitable (e.g. landmark-based) directions where appropriate.

3.2.3 Route and Direction Generation

Elliot and Lesk studied how to replicate human direction giving abilities [21]. The authors observed that humans followed a guided depth-first search strategy, and they were able to generate similar paths through the use of a heuristically guided depth first search penalizing turning. Duckham and Kulik created a system for generating “simplest paths” based on a metric for evaluating the cost of different turn actions (turning at a T-intersection is typically better than turning at a 4-way intersection) and finding the path that minimizes this cost [20]. This technique was further modified to deal with complex intersections, avoiding such areas while attempting to keep path lengths short [34].

Richter and Klippel defined a strategy for generating what they call “context-specific route directions,” directions which use knowledge of the structure of the environment to come up with easy-to-follow directions [86], later implementing this in the form of GUARD, a method for generating such directions [84]. GUARD considers turn distinctions (right vs. slight right) as well as landmarks in the environment to describe a given path in the best way possible. Later, Richter and Duckham created a method for generating the “simplest instructions” [85]. This method combines Richter’s GUARD method with cost metrics used by Duckham and Kulik in computing simple paths to create more effective sets of directions. In doing so, the technique not only finds a simple path through the environment, but also considers how the description of that path will affects followability.

With the exception of the "simplest instructions" algorithm, the main weakness of the previous work is the separation of path selection from path description. These processes are *not* independent. For example, the "simplest path" may still not be as easy to describe as a path with a few more turns but many landmarks along the way.

Richter and Duckham address this concern, incorporating additional cost metrics to take into account what they refer to as the *cognitive cost* of actions. As a result, both the simplicity of the path as well as the ease of interpreting directions along that path are considered when choosing directions. However, the algorithm does not attempt to deal with ambiguous descriptions of the environment. Our proposed method replaces the concept of cognitive cost with weight based on expected success rates for following given directions. As a result, properties such as environmental ambiguity are automatically incorporated into the weights.

3.3 Incorporating Wayfinder Models Into Planning

Evaluating successful arrival at the goal involves not only understanding how well people are able to follow the specified path, but how well they are able to recover when they deviate from it. Using a generative model offers a principled approach to evaluating the cost of an action. Given a set of directions, we can sample from our model (representing the distribution $p(x|d)$, or wayfinder position given a direction) and observe the resulting trajectories. From these observations, we gain situation-specific knowledge about the effectiveness of particular directions in our environment.

We formulate the direction finding task as an optimization problem. We assume that we are given a model m consisting of a map of the world and generative model that allows us to predict wayfinder actions for certain instructions. Then, assuming the wayfinder starts at position x_{start} and ends at x_{final}, our goal is to select a set of directions d maximizing the probability that the wayfinder reaches their goal:

$$\arg\max_{d} p(x_{\text{final}} = \text{destination}|d, m) \tag{3.1}$$

Particle simulation has proven to be a useful tool for navigation problems [101]. We employ it here to estimate the distribution of wayfinder positions given a set of directions. A naïve algorithm to find the best directions could generate every possible set of directions and run particle simulation for each set. The best directions would be the set for which the most particles arrived at the goal.

Fig. 3.1 shows an example of how such a technique can take advantage of a model to pick the best directions. In this environment, we are attempting to navigate from the black square in the

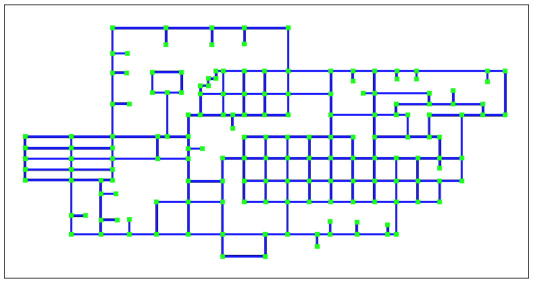


Figure 3.2: A hand-constructed map for simulated testing. This map covers an area roughly 11 km × 5 km and contains 167 nodes and 135 landmarks. Landmarks were generated and placed randomly from a family of 15 unique landmarks.

lower left to the black star in the upper right. The last two directions of a plan can be described as "Go until you stop at a T-intersection and turn left," and then "Go until you reach a dead end and you will have arrived at your destination," regardless of which of several horizontal crossroads you pick.

Fig. 3.1a shows the paths followed by simulated wayfinders for a set of directions following a shortest path. By chance, this happens to guide us to one of the aforementioned crossroads. However, due to the distance traveled to this crossroad and lack of good landmarks, many of our particles get lost.

A more complete search of the space based on our model turns up the directions highlighted by Fig. 3.1c. Many of the simulated particles make mistakes (in fact, the percentage of simulated wayfinders following every direction perfectly does not vary greatly between the two plans), but knowledge provided by our model allows us to position our plan among several acceptable crossroads so that erring wayfinders that turn late are more likely to turn onto a road that still allows them to correctly execute the later instructions. As a result, twice as many particles are able to navigate to the goal in Fig. 3.1c as in Fig. 3.1a.

3.3.1 Simulation and Model Formulation

We employ simulations, which allow us to rapidly evaluate the effects of employing different models as well as greatly accelerate and simplify the data collection process. To facilitate rapid implementation and testing, test environments were limited to planar graphs with only right angle turns (see Fig. 3.2 for a sample instance). For evaluation, three main environments were created by hand and randomly populated with landmarks, as well as a set of four environments used for collecting additional timing data. We assume that landmarks only affect one known decision point and that the direction from which we approach the intersection does not affect the usefulness of the landmark. As results were consistent across environments, we limit the presentation of results in this paper to the map seen in Fig. 3.2.

We employed four different categories of directions that our modeled wayfinder can understand. These were:

- METRIC: Go X m and turn
- INTERSECTION: Go to the X^{th} intersection and turn
- LANDMARK: Turn when you see *landmark* X
- GO UNTIL: Go in this direction until forced to make a choice and turn

We created a simple model based on previous literature in which humans become increasingly poor at counting intersections/measuring distances as the numbers in question become large. Conversely, LANDMARK and GO UNTIL directions were modeled as remaining robust regardless of distance, as previous research indicates that people are much better at following these types of instructions. We also represent the limitations of human memory with a fixed probability for which any given direction might be "forgotten," implicitly penalizing longer direction sets as being more difficult to follow.

To ensure that the directions generated were actually influenced by the model, we repeated tests with perturbed variations of the model. We expect, for example, that dramatically reducing the quality of LANDMARK directions should result in plans using them less often. Further discussion and results from these tests will be presented in Sec. 3.4.2. We also present results on the arrival rates of the wayfinder with varying direction-giving strategies. These numbers are not presented to show that our model is the state-of-the-art (it is not), but rather that the addition of a model to planning *improves* wayfinding performance.

3.3.2 Finding Directions Maximizing Arrival Rate

In this section, we will describe an ideal approach to selecting the set of directions maximizing wayfinder arrival rate, discuss optimization strategies for dealing with scalability issues and, finding such optimizations insufficient beyond small toy examples, present a method for arriving at paths that, while not guaranteed to be optimal, still offer improvement over shortest path directions.

To find the most effective set of directions for getting the wayfinder to the goal, we want to search through the space of possible directions, estimating the probability of reaching our goal using particles. Searching through this space is challenging, though, given the scale of the problem. The number of decisions to make from any given intersection is generally larger than just the node degree $\times$ the number of possible direction types. Directions do not merely describe how to get to immediately adjacent neighboring intersections, but also how to get to locations many intersections

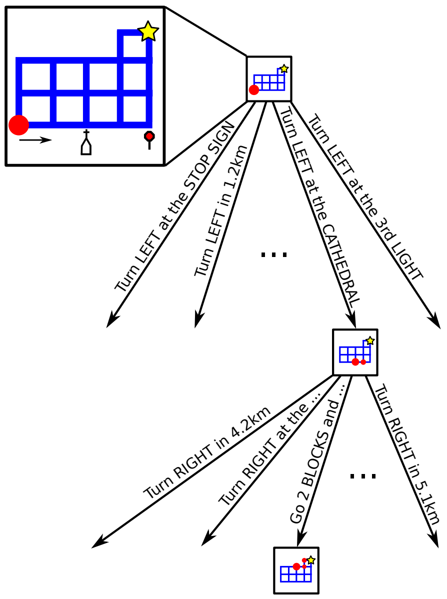


Figure 3.3: A small portion of a depth-first search through direction space. The distribution of particles through the environment is shown at several of the decision steps as red circles, with larger radii corresponding to higher particle densities.

away. Thus, even though our map is a simple planar graph, in reality, our planning takes place on a more complex multigraph where distant intersections are connected by edges describing a single direction that guides the wayfinder between them.

To illustrate the problem, consider the map seen in Fig. 3.2. The average branching factory on the underlying multigraph is 8.3 *before* accounting for the multiplicative increase caused by factoring in differing direction types. The map seen in Fig. 3.1 is even worse, owing to its grid-like structure, with an average branching factor of 10.9 before accounting for direction types. Even for the fairly simple model we employ, this means at least an additional 2× increase in average branching factor.

This large branching factor makes breadth-first searches (BFS) intractable, particularly in regards to memory usage, so a more directed approach is necessary. Though many choices are available at each decision point, it intuitively seems like this space could be pruned to make the problem tractable at smaller scales. Therefore, an optimized depth-first search (DFS) is employed when searching for the best set of directions. These optimizations are discussed in further detail below.

Fig. 3.3 illustrates a portion of our DFS through the multigraph. We employ branch and bound techniques to speed up the search through the map. The challenge here lies in determining an upper bound on path quality. At first glance, it would seem that deviation from the directed route could offer some bound on performance. By scoring plans based on the number of particles that may still reach the goal, we may discard plans with lower maximum success rates than our current best solution. However, by pruning out plans with many particles deviating from the directed route, we do not give these plans a chance to demonstrate good error-recovery characteristics. The problem, in essence, is that it is difficult to say when a particle is "lost" as opposed to one that will eventually work its way back on course. We do not want to throw away plans that may have wayward particles get back on track.

As a result, our DFS pruning strategy has a very conservative definition of being lost. Particles only become lost when they get caught in a situation where they cannot execute the desired instruction. When a particle encounters such a situation, it is marked "lost" and the score of the associated plan is decreased accordingly. Our bound for pruning, then, is the score of the best plan encountered to date.

For many plans, however, this still does not offer a useful performance improvement. Often the DFS will initially search down deep paths, resulting in poor initial plans and accordingly weak lower bounds. In these situations, pruning offers very little benefit. Thus, we explore several optimizations to speed up the search.

One such optimization is finding tighter initial bound early in the search process. An initial bound can be calculated by finding the shortest path to the goal and evaluating directions following that path. In a similar manner, when choosing the next branch to traverse, we select branches leading us physically closer to the goal, first. As a result, initial plans tend to converge onto the goal more quickly and, and a result, bound the results earlier. Especially in maps with grid-like structure, this is a very effective technique. However, in some cases, this results in the early pursuit of very bad paths, causing us to explore a second style of optimization.

Our second optimization is to use iterative deepening with DFS. Normal iterative deepening searches base their depth on the number of search steps taken. In this context, however, the physical depth of the search does not necessarily have any meaning. A plan with more directions might be more effective than one with fewer, so we cannot ignore plans of greater actual search depth. We can bound our search depth by minimum acceptable arrival rate, though. By artificially setting a lower bound for the search, we can constrain our search to only good plans, and if no such high quality plans are found, lower our standards progressively until a good plan is found. For plans with high arrival rates, this can lead to much faster convergence on a solution. However, as we go deeper into the search space, the cost of recomputing the same plans over and over adds up, eventually leading to a loss in performance. In testing, we found that enough queries benefited from iteratively deepening the search that these gains made up for the repeated computation performed after increasing the search depth.

The resulting structure of our algorithm is as follows:

1. Initialize a "plan" with many particles and an empty set of directions and add it to a stack.

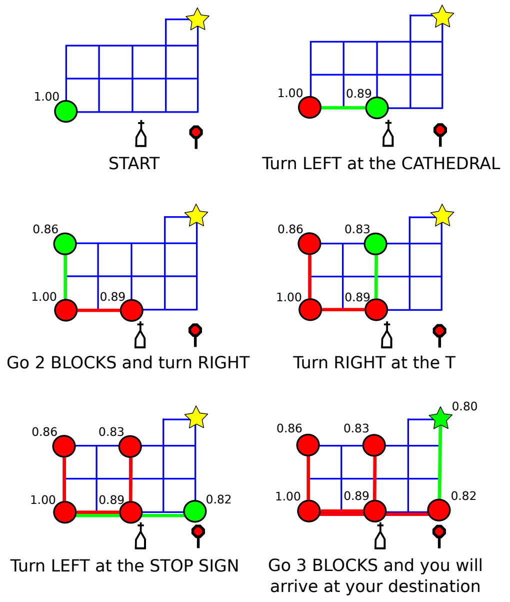

1.00
START
1.00
0.89
Turn LEFT at the CATHEDRAL
0.86
1.00
0.89
Go 2 BLOCKS and turn RIGHT
0.86
0.83
1.00
0.89
Turn RIGHT at the T
0.86
0.83
1.00
0.89
0.82
Turn LEFT at the STOP SIGN
0.80
0.86
0.83
1.00
0.89
0.82
Go 3 BLOCKS and you will arrive at your destination

Figure 3.4: An small example of a DART search. At each step, DART expands the node with the largest number of on-track particles (depicted in green) and adds it to a closed-list (depicted in red). When the goal node is expanded, the search ends.

2. While the stack still has entries, pull off the top entry and consider all possible, non-backtracking directions that can be taken from the current position as defined by the model.

3. Make copies of the plan for each possible direction and simulate the associated particles following that direction. Now the particles form some new distribution of positions and states in the world.

4. Sort the plans by distance of theoretical position to the goal and, if they do not violate our "depth" constraint, add them to the stack.

5. If iteratively deepening, repeat until a plan is found, increasing the search depth as necessary.

The asymptotic runtime is impossible to fully assess without specific knowledge of the model. Runtime increases proportionately with the number of particles simulated, but the depth of the search is dependent on the structure of the map and the wayfinder's abilities. However, if we assume that the maximum search depth is d, then for number of particles P and branching factor b, the worst-case asymptotic runtime of our DFS is $\mathcal{O}(Pb^d)$. As will be discussed in Sec. 3.4, we found that, even with our optimizations, the search times for non-trivial sets of directions were too long to be of use, indicating that our pruning is unable to make a substantial enough dent in the search space.

3.3.3 Direction Approximation through Random Trials

Since we cannot always afford to compute the best possible set of directions for an environment with our DFS method, we instead seek to quickly approximate the best directions while still making use of our model. One of the main difficulties with optimizing the depth-first search is that plan quality does not decrease monotonically as the search progresses. Though a particle may not be on the directed route at a given step, this does not preclude it getting back on course and successfully reaching the goal. As a result, we are very conservative when eliminating potentials plans from consideration. For DART, we take a more aggressive approach, treating particles that have wandered off course as irrecoverable when evaluating current plan quality. We make this decision based on the belief that situations in which the environmental structure allows particles to recover are rare, and thus the losses in overall plan quality are negligible. Based on these assumptions, we may now build a much more rapid search.

Given our assumption, the optimization problem is greatly simplified. While we are still maximizing (3.1), $p(x_{\text{final}} = \text{destination}|d, m)$ may now be expressed as:

$$p(x_{\text{final}} = \text{destination}|d, m) \approx \prod_{i=0}^{|d|} p(d_i|m, d_{i-1}) \tag{3.2}$$

Where $p(d_i|m, d_{i-1})$ is the probability of executing the i^{th} instruction successfully. The success of a plan, then, is based strictly on the success of executing each individual direction in the plan successfully. We may make this claim because we expect each additional instruction to at best leave the probability of the wayfinder reaching unchanged and at worst decrease it. Therefore, actions may be considered in sequence. This means our optimization problem looks like:

$$\arg\max_{d} \prod_{i=0}^{|d|} p(d_i|m, d_{i-1}) \tag{3.3}$$

Now the problem resembles a traditional shortest-path graph algorithm, in this case with edge weights equal to the probability of following the next direction. To find the optimal path to all destinations from a given start point, we apply a version of Dijkstra's single-source shortest path algorithm [18] to find the best set of directions to any given point. A short example of the resulting search process can be seen in Fig. 3.4.

We track potential plans in a priority queue, keeping plans with more particles that have correctly followed the directions so far at the top of the queue. During the expansion calls, we generate the copies of the plan, including the state of all of the current particles, for each possible next direction and simulate taking that step before adding the new plans to the priority queue. A straightforward implementation of this algorithm performs in $\mathcal{O}(|P||D||E| + |V| \log |V|)$, where

$|P|$ is the number of particles to simulate, $|D|$ is the number of possible direction types, $|E|$ is the number of edges in our graph of the world, and $|V|$ is the number of vertices in the graph.

The solutions returned by DART fall into a similar category as those computed by Richter and Duckham [85]. DART computes the path through the environment that will be followed the most reliably, ignoring possible advantages in ambiguous descriptions. While the algorithm *may* still benefit occasionally from particles getting back on course, the algorithm makes no guarantees that it will find such paths if they exist. While DART does not possess the full descriptive power of the DFS, it has the practical advantage of returning a "good" set of directions in a consistently low amount of time, regardless of the complexity of the directions necessary to navigate between start and goal.

3.4 Results

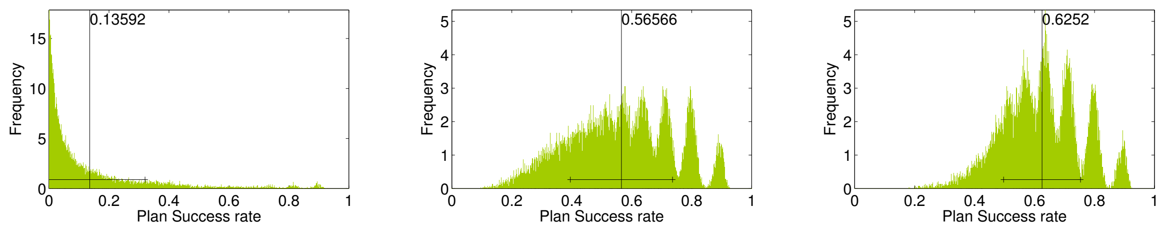
To evaluate our technique, we constructed a simple simulation environment where we could rapidly construct maps and populate them with random landmarks. This environment has been described in Section 3.3.1. Our simulations were performed with 1000 particles per plan unless otherwise noted.

3.4.1 DART vs. Shortest-Path plans

Our primary metric for evaluation is the successful arrival rate of our simulated particles. Higher percentages clearly imply better sets of directions according to the model, since more particles were able to successfully execute the plan. The "forgetfulness factor" acts to penalize longer plans, and for our parameter settings results in periodic peaks in our distribution of direction quality corresponding with the number of directions given. As a baseline, we compared directions generated based on our wayfinder model to the standard of the day: shortest-path based METRIC directions. Since it is commonly accepted in the literature that fewer turns are preferable to many, we choose the shortest path with the fewest turns when evaluating.

Our first use of the model was to come up with the best directions to follow the same shortest-path described by the metric-only plan. Given knowledge of the model, the quality of the plan should increase or, if metric-only directions are best, remain unchanged. Second, we compute directions using DART, searching beyond the bounds of the shortest-path to see if plan quality could potentially be improved.

For every start/goal pair in the environment pictured in Fig 3.2, we computed a plan using each of these three techniques. The resulting distributions of success rates can be seen in Fig 3.5. As expected, METRIC-only directions proved hard to follow for our modeled wayfinder. Model-based



(a) METRIC-only shortest-path (b) Model-optimized shortest-path (c) DART

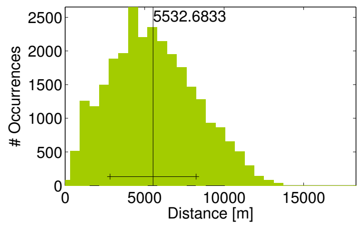
Figure 3.5: Histograms of success rates for METRIC-only shortest-path, model-optimized shortest path, and DART best path directions for every pair of nodes in the map pictured in Fig. 3.2. Plans using only METRIC directions deteriorate rapidly as path length and complexity increases, yielding poor wayfinder arrival rates. By using the model to choose the best direction for navigating between each turn on the shortest-path route, results were dramatically improved. DART further improves on the optimized shortest-path results by deviating from the shortest path to improve plan clarity. The mean of each distribution is denoted by a vertical black line, and bars extend out from the mean to indicate one standard deviation.

shortest-path directions proved significantly easier to follow. DART, however, was able to find the most effective directions at getting the wayfinder to the goal.

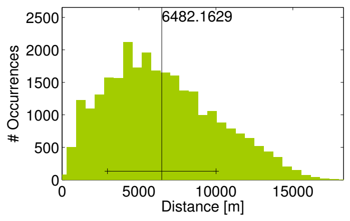
The distribution of wayfinder arrival rates shifts noticeably forward between DART and model-optimized shortest-path directions. In particular, DART improves upon the lowest-quality plans for model-optimized shortest paths, indicating that DART is better at directing wayfinders to faraway points where shortest-paths may become quite complex. While DART also boosts the number of high-arrival-rate plans, it is unsurprising that there is less change at the upper end of the performance spectrum. High quality shortest-path plans tend to consist of very few instructions and, as a result, offer little room for improvement.

The downside to DART is that, be definition, the routes it selects are of equal length or longer than those selected by the shortest-path strategies. Since our previous tests show that the shortest path is not always the best, we expect, then, that the paths selected by DART with will longer. To gain a better picture into how much we deviate from the shortest path, we also collected travel distances for our three evaluation methods. These can be seen in Fig. 3.6. As expected, the large improvements in arrival rate come at the price of distance traveled. The worst-case increase between plan distances is roughly 20%.

What is important to remember is that distance metrics are only important in terms of wayfinders who *successfully* navigate the route. A short route with a low probability of successful navigation will, in practice, have a longer expected travel distance as the wayfinder will need to navigate extra distance to get back on course.



(a) Shortest-path



(b) DART

Figure 3.6: Histogram of length of directed routes for shortest-path directions vs. DART best directions. Though DART is able to outperform shortest-path best directions in terms of wayfinder rate-of-arrival, it can pay a noticeable price in the distance traveled to the reach the goal. Our model places great weight on less distance-centric direction types such as landmark reliance, and thus is more willing to travel more out of its way for landmarks simplifying the overall path. The mean of each distribution is denoted by a vertical black line, and bars extend out from the mean to indicate one standard deviation.

3.4.2 Effects of Model Variation

We perform another test in which we perturb the parameter settings of our original model for two reasons. The first is to verify that our improvement over METRIC-only directions is not solely due to our model excessively penalizing wayfinders following these directions. The second, more important reason is to show that our algorithm is able to successfully adapt to different models. To accomplish these, we performed a second test with a model greatly penalizing the wayfinder's ability to recognize landmarks while improving intersection counting and distance tracking abilities. As a result, we expect the improved performance across METRIC and INTERSECTION directions to be reflected in the types of directions chosen. Further, the performance of METRIC-only shortest-path directions should improve.

We model METRIC direction following capabilities with a Gaussian centered at the requested direction distance. The variance of this Gaussian grows as the mean grows larger, following the form:

$$\sigma^2 = \frac{\mu}{k}\log(\mu + 1) \tag{3.4}$$

where μ is measured in kilometers and k is a parameter.

Our METRIC model is loosely based on the literature stating that humans have trouble with large numbers [95]. The leading μ term is multiplied by a $\log$ term representing a slow-percentage of the distance requested. The k term scales this percentage. We chose to model the proportional increase as slowly growing instead of linearly proportional based off of the intuition that, when traveling much longer distances, it becomes increasingly difficult to remember exactly how far one has traveled, leading to further errors. The Gaussian was selected for simplicity.

For the results already presented, we used the parameter $k = 200$, meaning the large majority of wayfinders traveling 1 km will travel a distance within roughly 0.1 km of this amount. To improve the quality of METRIC directions in our second model, we set $k = 1000$, as a result roughly halving the expected error in metric directions. As seen in Fig. 3.7a, around half of the possible directions in our testing space fall at or under the 1 km mark, so the majority of METRIC directions considered will be followed fairly accurately.

Fig. 3.7b shows the resulting distribution of directions types for our original model vs. the heavily modified version. As can be seen, the penalized landmark type directions completely disappear in favor of the now more reliable INTERSECTION and METRIC directions. METRIC directions, in particular, benefit from a vast increase in wayfinder following ability. This highlights our method's ability to successfully adapt to varying models as well as improve greatly on the standard practice of only issuing METRIC directions along the shortest path.

Fig. 3.8 shows the success rate distributions based on our perturbed model. Greatly enhancing the reliability of METRIC directions noticeably helps the metric-only shortest-path plans, but these solutions still lag behind the model-optimized shortest-path and DART best directions. DART still computes the best plans of the three, and with a very tight distribution of success rates.

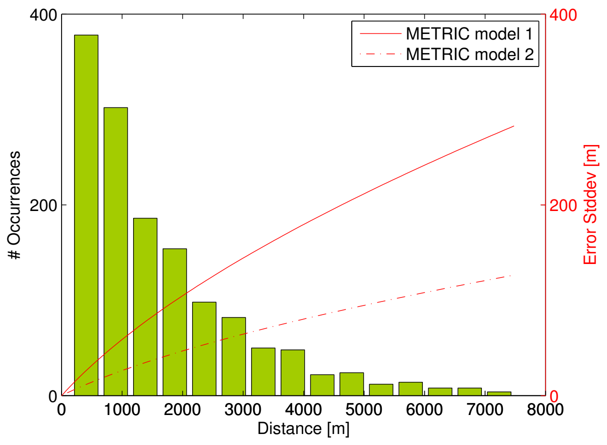
3.4.3 DFS vs. DART

Though the full DFS described in Sec. 3.3.2 can use to model create directions that more-easily recover from error, we were interested in seeing how much this actually matters. Attempts to collect data comparing DART to DFS at the same scale as our DART vs. shortest-path data sets proved computationally expensive for 1000 particles. Instead, we tested the two on the small environment seen in Fig. 3.1. The environment was designed to offer many routes of similar structure between various points to give the DFS as much an edge as possible.

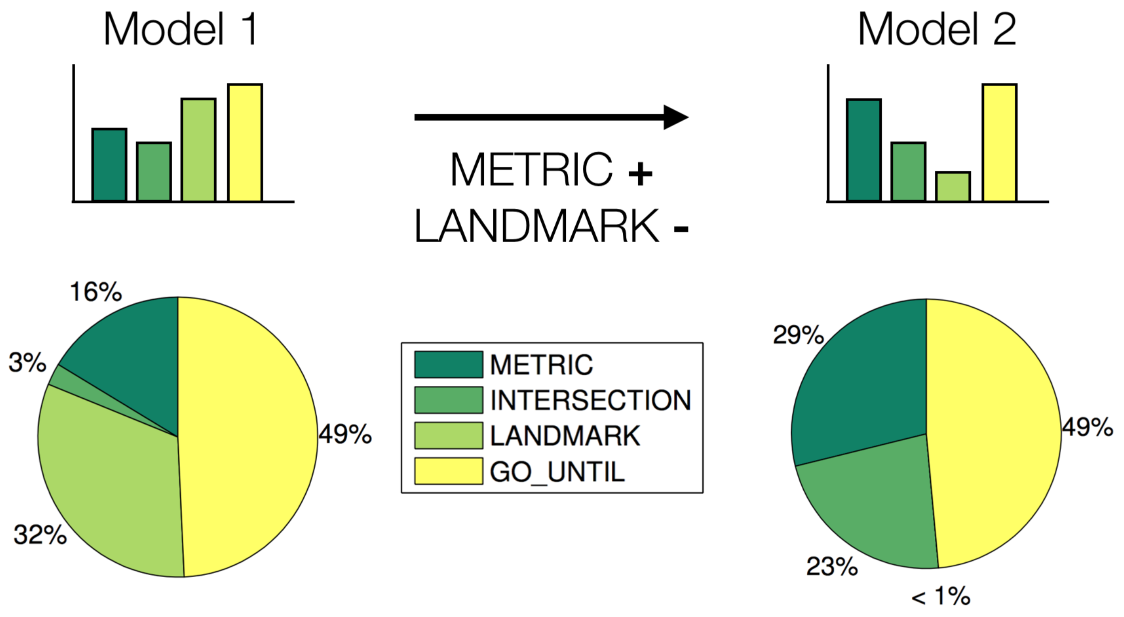
Even at this small scale, DFS of $\sim$1600 examples was time consuming, taking 17.7 hrs to compute, or an average of 40 s/plan compared to 0.5 s/plan for DART. As expected, DFS was able to find some routes dramatically increasing arrival rates at the goal compared to DART. Typically, however, DART was able to find equivalent or near-equivalent routes to DFS. This suggests that generally, one set of directions is so clearly the best that the benefit of baking error recovery into the plan is minimal.

3.4.4 The Effect of Particle Population on Performance

It is desireable to determine how many particles to use in simulation. We would like to balance the speed of algorithmic performance against the accuracy of our representations of the distributions of directions. To determine this value, we consider two metrics: the time is takes to return a query and

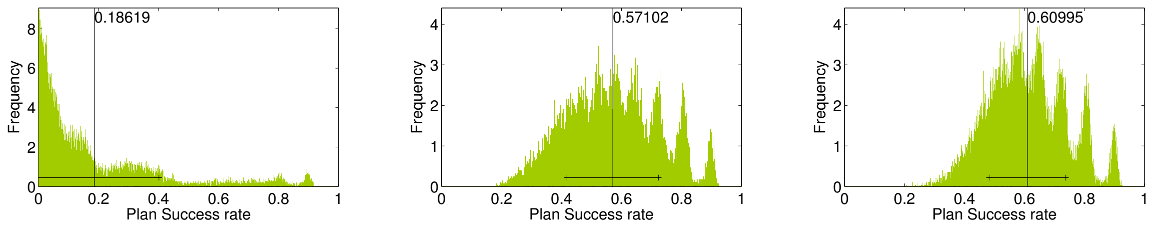


(a) Distribution of METRIC Directions vs. Expected Error in METRIC Direction Following



(b) Distribution of Direction Types Used for Varying Models

Figure 3.7: The effect of changes in METRIC direction following capabilities on the direction types selected when DART generates directions. The distribution of possible METRIC direction distances compared to the standard deviation in error for the two models used to collect data can be seen in (a). Over half of all possible METRIC directions in our test environment are of distances of 2 km or less, meaning for both models tested, we expect wayfinders to typically be within 100 m of the requested travel distance. As (b) shows, the distribution of direction types for the two models varied greatly when the model was altered. Thus, when the quality of METRIC and INTERSECTION directions was increased while LANDMARK recognition abilities were drastically decreased, DART responded by virtually eliminating all LANDMARK directions and by increasing the quantity of METRIC and INTERSECTION directions.



(a) Metric-only shortest-path (b) Model-optimized shortest-path (c) DART

Figure 3.8: Histograms of successful arrival rates for several route description strategies after perturbing the model. Even with a wayfinder modeled with greatly improved abilities to follow METRIC directions, the shortest-path direction limited to METRIC directions only could not compete with the model-optimized or DART directions. Again, DART return the sets of directions with the best mean success rate.

the quality of the solution. We expect quality to increase with particle quantity while performance speed will suffer.

We performed trials for 100, 500, 1000, and 5000 particles on 4 maps of gradually increasing complexity. The maps in question had 29, 55, 98, and 167 decision points, respectively, and branching factors ranging from 8.4 to 11.9 before considering direction type.

For each map/particle number combination, we ran 50 planning trials on each of 50 randomly chosen start/goal pairs and computed the mean planning time as well as the mean success rate across trials. Returns in plan quality for additional particles diminish rapidly, showing almost no change when increasing the number of particles from 1000 to 5000 particles. However, computation time increases linearly in the number of particles. We find that 1000 particles strike the best balance of estimating the distribution of outcomes accurately while remaining fast (sub-second) for our test environments.

3.5 Conclusions

We formulated a strategy for construction reliable verbal directions given a black-box generative model of a wayfinder. We developed two particle simulation techniques for determining the best set of directions between two points, given this model. Such particle simulations are able to find directions through the environment that utilize knowledge of their surroundings such as landmarks in addition to harnessing knowledge about the ways in which the wayfinder will err to pick directions facilitating error recovery, when appropriate. Our preliminary results show that, even given very simple models and minimal knowledge of the environment, particle methods offer an improvement over currently-employed shortest-path methods.

Unfortunately, until the cost of full particle-based searches through the environment can be limited, such methods will have difficulty scaling to real-world environments. Thus, methods approximating ideal results such as DART must be employed for applications where near real-time performance is necessary. Systems deployed with such direction-giving capabilities could play an important role in assisting people through confusing, high-traffic areas such as hospitals or busy tourist locations as well as in currently existing vehicular and phone-based navigation contexts. These methods could also be applied to robotic navigation tasks, which is discussed in more detail in the next chapter.

CHAPTER 4

Robot Navigation with Policy Chains

4.1 Introduction

In the previous chapter, we showed that, given a topological map of the environment and a generative model of a wayfinder, we can compute sets of actions to take to robustly guide the wayfinder through the environment. In this chapter, we describe the process implementing this style of planning for a robotic system.

We present a method for generating plans in the form of chains of successively executed control sub-policies, an example of which can be seen in Fig. 4.1. Sub-policies take the form of closed-loop control laws such as wall following, visual servoing, or goal pursuit. Sub-policies are switched between based on conditions such as observing landmarks in the environment. We assume the existence of a sensor model approximating the robot, as well as a map of the environment. By simulating repeated, noisy executions of actions available to the robot, we sample from the posterior distribution of possible robot locations. This allows us to find policies that maximize the expected rewards.

Our contributions in this chapter are:

- A novel planning framework in which plans are expressed as a chain of <sub-policy, termination condition> tuples.
- The identification of a set of useful, general-purpose policies.
- We show that this framework can take plan robustness into account, selecting routes that will avoid sensing failures.
- We validate our simulated results against a real-world scenario.

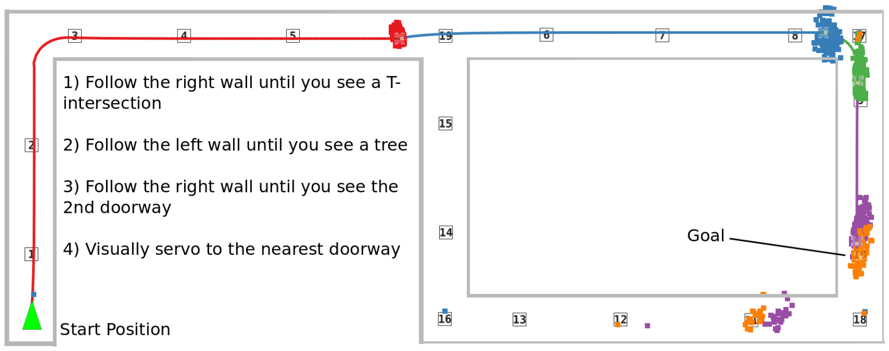

1) Follow the right wall until you see a T-intersection
2) Follow the left wall until you see a tree
3) Follow the right wall until you see the 2nd doorway
4) Visually servo to the nearest doorway
Start Position
Goal
1
2
3
4
5
19
6
7
8
15
14
16
13
12
18

Figure 4.1: The expected trajectory when executing a policy-chain plan compared to open-loop odometry estimates of 250 simulated executions of the policy-chain. Because the plan instructs the robot to wall follow between reliably detected landmarks instead of relying on its noisy pose estimates for positioning, the impact of odometry noise is mitigated.

4.2 Related Work

The most straightforward techniques guide a robot towards its goal via potential fields. The fields generally slope towards the goal and are constructed such that the robot will also avoid obstacles [58]. Potential fields are well suited to dynamic environments, as little additional work is needed to handle moving obstacles, but most such methods are susceptible to local minima [41, 28].

Trajectory following methods allow plans to be constructed that avoid local minima and other hazards. Navigation occurs in two phases. First, a safe trajectory is computed from the current robot pose to the goal pose in a planning phase. Oftentimes, the goal is to find the shortest possible path. The trajectory might be as simple as a Manhattan path, but for systems where dynamics are important, the trajectory may specify velocities, orientations, etc. that must be met at various points of execution [19]. Plans may even attempt to incorporate the avoidance of dynamic obstacles [25, 3].

After the plan is computed, a trajectory follower takes over, attempting to guide the robot along the specified path [90, 4, 57]. In dynamic environments, replanning may be necessary to handle unexpected obstacles. The D* family of algorithms and others incorporate incremental correction steps to aid in recovery from such complications [93, 24, 61].

The above navigation strategies are dependent on good localization, but real-world systems must operate with uncertain estimates of their pose. This has lead to examination of the problem of planning to minimize uncertainty. Since planning now takes into account the possibility of noisy execution, plans are selected to maximize the belief that the robot will reach the goal. These techniques are complementary to our objective, insofar as they explore the problem of choosing plans based on desired levels of system performance.

A popular strategy is to model the robot belief space during navigation using a POMDP. However, while POMDPs provide a rigorous framework with which to model the problem of robotic navigation, the complexity of the belief space and long time horizon often involved in robotic planning typically render exact solutions intractable. Even the best state-of-the-art solvers can take hours to solve fairly small real-world problems [55]. As a result, recent efforts have focused on approximate or sample-based solutions. For example, Kurniawati et al. generate a simplified roadmap through the robot's state space by reducing the space to representative sampled milestones [54]. This allows them to solve previously intractable problems in minutes, though the domains are still fairly small.

Candido and Hutchinson take an approach similar to the one proposed in this paper, showing that POMDP-style planning can be used to choose a set of local control policies and switching conditions that attempt to minimize positional uncertainty at the end of plan execution [14]. Their algorithm differs from ours in that they optimize only across switching conditions, alternating between sub-policies pursuing the goal state and sub-policies attempting to minimize entropy.

In a slightly different approach, Prentice and Roy introduce the Belief Roadmap [81], a variant of the Probabalistic Roadmap algorithm that incorporates belief about pose into the planning process. Using the Belief Roadmap, robots may compute plans reducing belief uncertainty, resulting in improved positional accuracy. In this way, they try to mitigate the impact of sensing uncertainty during navigation.

Traditional navigation techniques have proven effective in many real-world systems, yet their performance is limited by their dependence on precise localization and execution. Even systems designed to increase the robustness of execution do so by focusing on reducing localization uncertainty. While modern localization systems can be operated reliably for long periods, they are still subject to catastrophic failures, for example, due to incorrect data association. We propose that plan robustness can also benefit by incorporating policies that do not depend on continuous localization.

4.3 Navigating with Sub-Policies

Robots must plan to navigate reliably in the presence of sensor noise and imprecise localization. The most rigorous formulation of the search for such a plan is a POMDP. In its most straightforward form for this problem, the POMDP would consider the current pose of the robot and its observations and map these to the best possible left and right motor commands according to some reward function. Unfortunately, this is an impractical solution to the problem as the space of possible states and actions is immense [78].

Previous efforts have been made to make the problem more tractable, including the use of point-based solvers [79, 55] and by generating belief roadmaps to plan through [81]. Instead, we propose formulating the problem as a search for a *policy chain* composed of tuples of pre-constructed sub-policies and termination conditions. Sub-policies refer to closed-loop behaviors executable by the robot, such as wall or hall following, visual servoing, and greedy goal pursuit. The resulting plan is executed in a similar manner to a finite state machine (FSM), with the active policy switching to the next policy when the active termination condition is met.

A key challenge when selecting the optimal sequence of control policies is determining when and where different control strategies are appropriate. The structure in the environment and sensors available to the robot play a critical role in determining the effectiveness of different navigation techniques. Kuipers's SSH [52] presents a framework with which robots can reason about the space around them in support of navigation tasks. Critically, a *control level* of this hierarchy is proposed in which the robot builds representations of space based on the set of control policies available to the robot. Key "places" are designated by identifying locations in the environment in which the robot enters a stable limit cycle by following particular policies (such as wall following). In this manner, the robot can reason about chains of policies to follow between these key locations, enabling the construction of larger-scale topological representations of the environment. In this work, we propose extending the ideas behind the SSH's control level, varying not only policies, but also the conditions under which it is appropriate to switch between them. Decoupling the control policy from the termination condition gives the robot greater flexibility in finding solutions, though at an increase in the cost of planning.

Our approach retains one of the critical advantages of the SSH – that choosing a control policy (even with a termination condition) dramatically reduces the search space versus more metrical planning approaches. We also retain the notion that a control policy can be treated like a black box – from a planning perspective, the only relevant property of a policy is that it allows a robot to traverse from one place to another. Sub-policies also offer a pragmatic engineering approach: system designers can construct policies that are well-characterized and well-suited to the target environment. An example can be seen in Fig. 4.2. A robot trying to navigate through a doorway and down a hall could accomplish this by planning a specific trajectory, but the task may also be described more generally as a composition of several sub-policies.

$<$Sub-policy, termination condition$>$ tuples also offer the benefit of encoding high-level goal knowledge directly into the plan in a way that can aid in execution. For example, a policy to follow a hallway until a T-intersection is encountered eliminates the need for precise localization during execution, instead relying on the detection of the T-intersection to cue the robot to change sub-policies. Some controls policies can also help constrain our belief about pose estimates. A robot employing a wall following sub-policy is expected to have little variation in lateral pose, instead

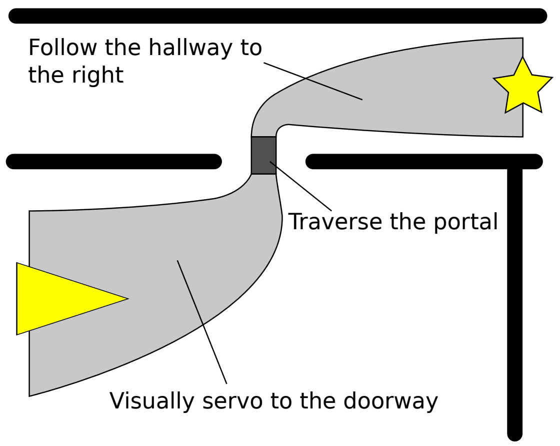

Follow the hallway to the right
Traverse the portal
Visually servo to the doorway

Figure 4.2: By navigating via sub-policies, the plan describes the task adequately for execution without over-constraining execution.

mapping its pose to the 1-dimensional manifold of position along the wall. Finally, policy-chains are straightforward to evaluate with forward simulation, allowing us to marginalize over stochastic variables such as noisy observations and treat tuples as single action steps. In this manner, we reduce the planning depth required to find a solution.

4.4 Implementation

We assume that we have already collected an accurate map of the world and that we begin localized in it. We also assume a sensor model that reasonably approximates the robot's error rates. For this work, switching policies are limited to landmark-based observations and checks that the robot has reached a stable state of execution, as when a hall follower reaches a dead end. This allows the model to be easily computed empirically by comparing sensor data against ground truth, with the locations of landmarks known in advance.

We evaluate plan utility by rewarding higher probabilities of successful plan execution and penalizing longer expected trajectories. The utility of state x_t is given as

$$R(x_t) = \lambda p - d \tag{4.1}$$

where d is the estimated distance the robot will travel executing this sequence of behaviors (including an estimate of how much distance remains), p is the estimated probability that the robot will end execution of the plan within an acceptable range of the goal, and λ may be tuned to weight the importance of arrival rate. If λ is set to 0, the planner simply searches for the shortest path to the goal constrained by the provided behaviors, while a λ value of 100 allows the robot to travel 1 m

Algorithm 1 Policy-chain A$*$

```
1: procedure POLICY-CHAIN-SEARCH($start, goal$)
2:     $heap \leftarrow start$
3:     while notEmpty($heap$) do
4:         $n \leftarrow$ best($heap$)
5:         if $n == goal$ then
6:             return $n$
7:         end if
8:         $child \leftarrow$ nextChild($n$)
9:         $heap \leftarrow$ simulate($child$)
10:        if hasMoreChildren($n$) then
11:            $heap \leftarrow n$
12:        end if
13:    end while
14:    return null
15: end procedure
```

further for every additional percent chance that it will successfully complete its task. The distance penalty acts as a discount factor, bounding the search space by diminishing the value of rewards gained far in the future. The choice of λ is left up to the system designer.

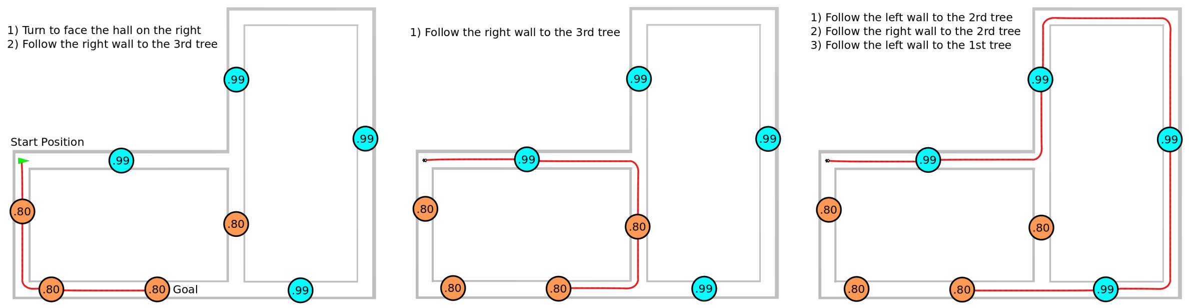
4.4.1 Choosing Sub-Policies

It is important to choose a rich enough set of sub-policies and termination conditions, as an incomplete set can leave regions of the environment unreachable. These choices are system- and task-dependent. In this work, we consider a mobile robot navigating an indoor environment composed of hallways and landmarks detectable by the robot. The set of sub-policies used consists of:

- Follow $\left[\text{left/right}\right]$ wall
- Visually servo towards $\left[\text{object class}\right]$
- Orient to face $\left[\text{heading}\right]$

Termination conditions are:

- See an $\left[\text{object class}\right]$
- No longer moving


1) Turn to face the hall on the right
2) Follow the right wall to the 3rd tree
Start Position
Goal
.99
.80
1) Follow the right wall to the 3rd tree
1) Follow the left wall to the 2rd tree
2) Follow the right wall to the 2rd tree
3) Follow the left wall to the 1st tree

(a) $\lambda = 0$ (b) $\lambda = 250$ (c) $\lambda = 1000$

Figure 4.3: Different values of λ can yield very different policy-chains. Several of the landmarks (orange) are misdetected 20% of the time, while the other landmarks (blue) are nearly perfectly detectable. As λ increases, the robot shifts its to longer plans to avoid depending on less reliable switching conditions.

When generating <sub-policy, termination condition> tuples, we assume landmarks will be correctly identified and that relative orientations can be computed such that the robot will be able to turn to face down a different hallway. We assume landmark observations to be independent of each other, allowing conditions of the form "See the n^{th} $\left[\text{object class}\right]$" to be treated as equivalent to executing a sequence of n single $\left[\text{object class}\right]$ sightings.

4.4.2 Building the Search Tree

Given a set of sub-policies and termination conditions, the next challenge is determining which pairings are appropriate. In the most naïve case, we would consider every possible pairing of sub-policy and termination condition at every step of the search. The search tree grows exponentially in the branching factor, so in practice, we wish to only generate feasibly executable actions.

In order to determine which termination conditions are likely to be relevant, we employ forward simulation. Proposals for child actions are generated via single noise-free simulations of sub-policies executed for a fixed time period. During these simulations, statistics are collected regarding which landmarks should be seen and when. Based on these statistics, we determine relevant pairings of landmark-based termination conditions with sub-policies. We also use these statistics to compute the best-case score for this action.

Unfortunately, it is unlikely that the robot will perfectly execute the proposed actions. Completely describing the ways in which the robot may imperfectly execute a sequence of actions is impractical, though, as the accounting for decision points results in exponential growth in the representation. Instead, Monte Carlo simulations are used to estimate the distribution of states that will result from executing a <sub-policy, termination condition> tuple. More simulations result in improved estimates, but at the cost of increased computation. The number of simulations sufficient for good distribution estimates is system dependent.

4.4.3 Searching for Policy-Chains

We employ an A* search strategy [35] with a lazy branch evaluation scheme to compute the optimal policy chain. Pseudocode is given in Algorithm 1. Proposed policy-chains are scored based on a modified version of our scoring function incorporating a heuristic estimating the distance remaining to the goal. We cheaply pre-compute these distances using the wavefront algorithm, allowing tighter estimates than Euclidean distance as well as fast lookup during search. Expected arrival rate is computed based on the running distribution of states sampled during Monte Carlo simulation. In a manner similar to the one employed in [31], we only count states which are "on-track" as being expected to arrive.

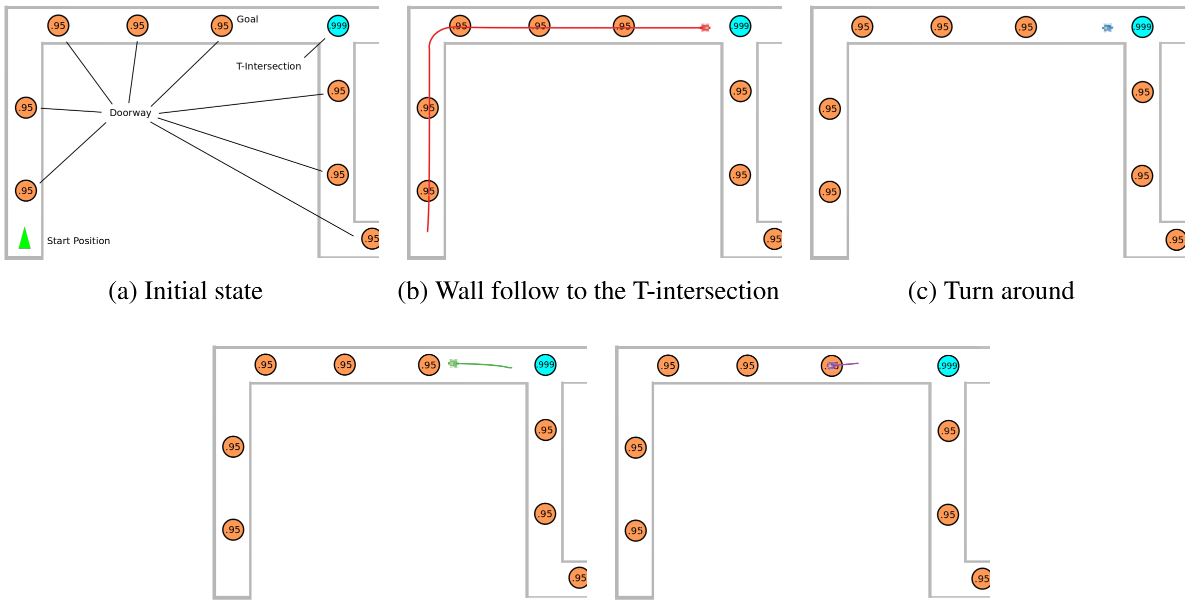
The dominant cost during search is the Monte Carlo simulation of sub-policies used to generate the distribution of new states. To avoid unnecessary computation, we use a lazy evaluation technique. We consult the best-case scores computed during child generation to get an early estimate about which actions will guide the search closer to the goal. The best-scoring child is greedily evaluated, after which it and its parent are added to the search heap.

If the child is the new best search option, search proceeds without paying the full evaluation cost for the other children. If the parent later reappears as the most promising avenue of search, it is pulled from the heap once more, at which point the next most promising child is evaluated, and so forth. In this manner, many searches are able to converge directly towards the goal without performing unnecessary simulations.

4.5 Results

In this section, we demonstrate that our method can generate correct policy-chains for navigation and examine the impact of varying noise and reward parameters. We test our method in simulation. The robot is modeled as differentially driven, with odometry sensing coming from encoders polluted with Gaussian noise. Additionally, the simulated robot is equipped with a LIDAR-style range sensor measuring ranges at 1 degree increments for a 180 degree field-of-view around the robot. These range measurements are also polluted with independently sampled Gaussian noise.

Landmarks are modeled as having a distribution of possible classification results, subject to some latching error. In other words, when a landmark is first observed, we query it for an identifying label. This label is randomly sampled from the distribution of possible labels for that landmark, at which point it is assigned to that landmark until it is out of view. It is also possible for a landmark to return a null label, indicating that the robot failed to detect the landmark entirely. Landmarks



(a) Initial state (b) Wall follow to the T-intersection (c) Turn around

(d) Wall follow until you see a door (e) Visually servo to the door

Figure 4.4: A sequence of images demonstrating the execution of a policy chain. It is often advantageous to drive further for the benefit of greater probability of arrival. For example, when navigating to the last doorway before the hall, it is easier to go to the end of the hall and back one doorway than to try to count every door along the way.

are also subject to noise regarding the range from which they are observed. This is intended to simulate the variability in performance often seen in real-world vision problems.

4.5.1 Avoiding Error-Prone Actions

We examine a simple world to demonstrate that the search adapts its behavior based on the value of λ. In the simulated world pictured in Fig. 4.3, we model a set of hallways in which several trees have been placed as decorations. Some of the trees are older and larger, and as a result, our object detector classifies them correctly 99% of the time. However, some of the trees are brand new. Because they are so small, sometimes our object detector confuses them for rubber potted plants. As a result, the robot only recognizes them correctly as trees 80% of the time.

Results of navigation for several settings of λ can be seen in Fig. 4.3. When $\lambda = 0$, no reward can be gained by finding more reliable routes to the goal. As a result, the shortest path is pursued. When we increase the value of λ sufficiently, though, the robot changes policies to deviate to more reliable, but longer, routes.

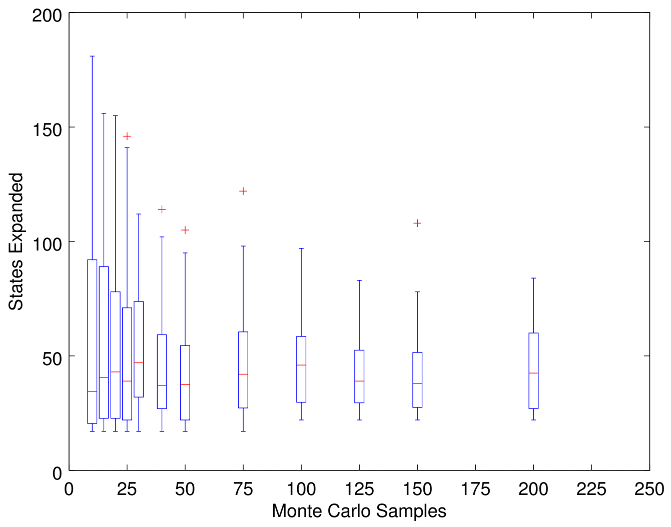


Figure 4.5: Distributions of the number of state expansions performed for varying quantities of Monte Carlo samples. As more simulations are performed, bounds on scores become tighter, reducing variation in the number of states expanded during search. Beyond a certain point, however, returns are limited, as computation cost grows linearly with the number of simulations performed during sampling.

4.5.2 When Longer Paths are Better

By incorporating reliability into planning, search may ensure that it uses both sufficiently repeatable sub-policies as well as sufficiently repeatable switching conditions. The resulting plans often resemble instructions similar to those one might give a human requesting directions.

Consider the example in Fig. 4.4. The robot would like to navigate to the fifth doorway along the hall and remind the occupant of that office that it's time for a meeting. Unfortunately, the robot's door detector is not fully reliable, so driving down the hall and counting five doorways will often result in the robot failing to count a door and overshooting its goal. Ultimately, the robot must correctly recognize at least one door (the one it must knock on) to successfully complete its task. However, the robot may use other structure in the environment to mitigate the problem.

In this case, the robot can reliably detect T-intersections. Additionally, the fifth door from the robot is also the last door before a T-intersection. Our search recognizes this fact and, instead of instructing the robot to count five doors, tells it to drive until it sees the T-intersection, turn around, and then count a single door before visually servoing to the target so it may knock. As a result, the robot avoids unnecessary door recognition tasks and improves the likelihood with which it will successfully knock on the correct door.

4.5.3 Monte Carlo Samples

The majority of the computation time in our algorithm is devoted to simulation during Monte Carlo sampling. Thus, is it important that simulations may be performed quickly. Our simulator is implemented in Java and data was collected on a machine containing 8GB of RAM and a quad core Intel i7-2600K CPU clocked at 3.4GHz. We can simulate 1 second of robot operation in approximately 0.0004 seconds of CPU time, or 2500 times faster than real time.

System designers can change the number of Monte Carlo simulations to perform for each evaluation step to trade off between accuracy in score estimates and computation time. We expect tighter score estimates to result in a more efficient search, as we will more reliably be able to prune bad search paths. To verify this result in our system, we compute the plan from the example described in Sec. 4.5.2 50 times and observe how the search is executed. As Fig. 4.5 shows, the deviation in node expansions, particularly in the upper quartiles, quickly drops as the number of simulations performed increases.

4.5.4 Caching Plans in Topological Maps

This chapter describes a method for computing policy chains based online planning, but even with fast simulation, this process is computationally expensive. In practice, it is useful to cache common, optimal routes through the environment, such as the ones seen in Fig. 4.6, in the form of a topological map. The resulting maps can be used in a similar manner to a Probabalistic Roadmap [45], using online planning methods to plan a path into the topology, but using the stored information in the topology to compute large-scale routes that might otherwise be expensive to produce in real-time.

The strategy for building the map is simple. To restrict the set of nodes in the map, we only consider places marked by landmarks (e.g. doors, intersections), which are easily detected by the robot:

1. For each landmark, simulate the set of available actions which reach other landmarks, limiting the search depth to a depth of 1.

2. Simulate a sufficient number of times to estimate the distribution of possible outcomes of taking those actions.

3. Assemble the set of actions and distributions into a topological sub-map describing ways to navigate from the current landmark to its neighbors.

4. Construct the full topology by taking the union of these local sub-maps.

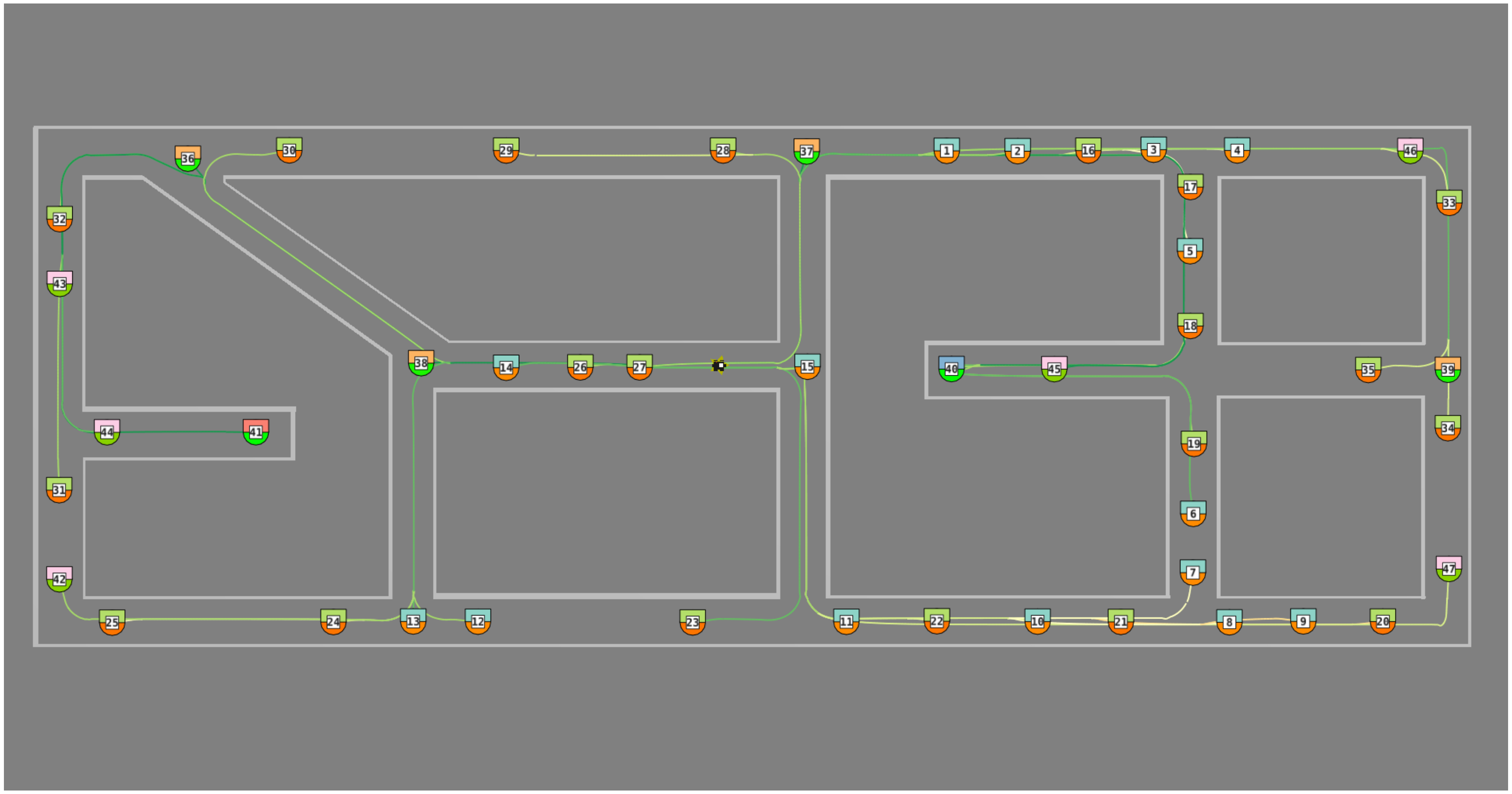


Figure 4.6: A minimum spanning tree (green) of actions leading from the current position of the robot (image center) to all possible landmarks (colorful shields). Important routes such as the ones pictured can be cached for future use by the robot, enabling more rapidly planning in the future.

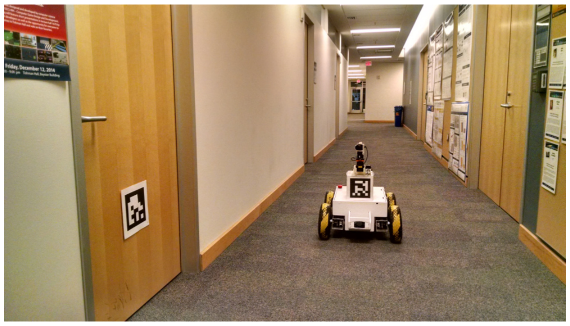


Figure 4.7: The robot platform used for real-world evaluation next to a doorway labeled with an AprilTag.

4.5.5 Real-world Results

We also validate policy-chain results in the real world. The hallways of our building were instrumented with AprilTag fiducials [74] as stand-ins for landmarks. By using reliably detectable fiducials, the system's vision performance may be degraded in a controlled fashion for testing. The testing platform was a custom robot equipped with LIDAR, an IMU, wheel encoders, and a camera was. The robot can be seen approaching an AprilTag in Fig. 4.7.

Two plans were constructed based on a model of the test environment: one with $\lambda = 0$, resulting in a shortest-path policy-chain, and the other with $\lambda = 1000$, which resulted in longer, but more reliable, sequence of actions to execute. The robot was tasked to execute the resulting plans 10 times each. We found real-world robot results to be similar to those seen in simulation. When following the shortest-path policy, which was based on counting unreliably detectable landmarks, the robot frequently missed a single observation, causing it to overshoot its destination. Conversely, the policy-chain optimized for robust execution more consistently guided the robot to its goal by reducing the number of failure-prone observations it depended on.

4.6 Conclusion

In this work, we have proposed a framework for the construction policy chains for navigation. Policy chains are composed of a series of <sub-policy, termination condition> tuples describing a sequence of closed-loop controls for the robot to execute. We use a best-first search to explore the space of actions executable by the robot and employ Monte Carlo simulations to evaluate expected performance in the presence of sensor errors. The resulting plans are optimized to balance successful plan execution against expected travel distance. We evaluated our system in simulation, comparing against real-world executions of plans to validate our results.

CHAPTER 5

Learning Switching Conditions [1]

5.1 Introduction

Landmarks or other forms of place recognition and provide valuable reference points in navigation tasks, providing localization cues to the robot. In DART, these features play a key role in determining the routes the robot will select. In the previous chapters, it was assumed that some set of place detectors exists, but not what form they would take. In this chapter, we address that assumption, proposing that structure-based cues are well-suited for DART's requirements. To detect these cues, we propose a method for semantic place classification, allowing the robot to recognize specific types of places and the transitions between them.

Environmental structure can encode key information about the type of place a robot is in. Corridors typically consist of long, narrow open spaces, whereas doorways appear as short, narrow gaps in structure. Robots can exploit this information to improve on or add to their capabilities. For example, knowing that a robot has transitioned from a room into a corridor adds extra context for a localization problem. By pairing semantic place knowledge with natural language, a robot can be given verbal instructions for navigation, such as "Follow this corridor until you reach an intersection, then take the hall on the left." Semantic information can also be used to segment the environment facilitating the creation of topological maps.

Given the ubiquity of range sensors in modern robotics, there is value in exploring methods for extracting as much information from them as possible. Though it is unlikely that 2D range data on its own is sufficient to reliably distinguish high level concepts such as room types, it has proven useful for more general place recognition tasks [8, 71], learning to distinguish between classes such as "doorway", "corridor" and "room." Much of the previous work in place recognition focuses on carefully hand-engineering features for the task at hand, often deriving them from 2D or 3D range data.

[1]© 2016. Adapted from Robert Goeddel and Edwin Olson, "Learning Semantic Place Labels from Occupancy Grids using CNNs," October 2016.

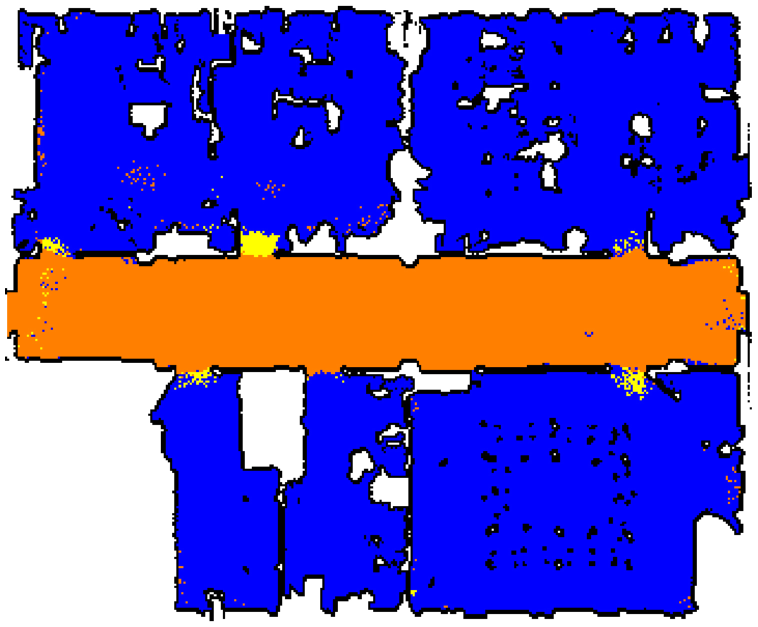


Figure 5.1: CNN-based classification results on the fr79 dataset. Blue denotes a room, orange a corridor, and yellow a doorway.

We propose that CNNs are particularly well suited to the task of place classification based on 2D range data. Range data is typically converted into occupancy grids, which are analogous to grayscale images depicting local structure. We hypothesize that, just as CNNs are adept at learning task appropriate features for classifying objects in images, they will be similarly adept at identifying features in occupancy grid "images" relevant to classifying place types. This approach can be used for general map annotation tasks, as seen in Fig. 5.1, and is also well suited for processing live data streams from robots, as might be desirable when following directions in an unknown environment.

In this work, we demonstrate that CNNs can, in fact, be used effectively to apply semantic labels to places based solely on 2D range data. In particular, on the classes "room" and "corridor", we obtain accuracy above 92%. We also analyze the strengths and weaknesses of the system. Our contributions in the work include:

- We propose and evaluate a specific network structure to improve performance with regards to inter-class confusion.
- We experimentally demonstrate that CNNs perform comparably to or better than previous work based on a well-known dataset.

5.2 Related Work

Semantic place categorization adds information about structure to robotic systems in addition to providing grounded place labels in aiding human interaction with robots. It has been of particular interest to the topological mapping community, since the identification and recognition of distinct places is critical to the functionality of topologically based systems. We broadly cluster approaches into two main groupings based on application: online labeling systems and map post-processing systems.

5.2.1 Online Labeling Systems

As the name implies, online labeling systems produce semantic place labels based on live sensor data from the robot. For example, one might combine range and camera data from a robot to attempt to classify the location the robot is currently in. These methods are well suited to tasks such as trajectory labeling or, more generally, operating in unknown environments.

For example, Mozos et al. present a place classification system using AdaBoost to distinguish places based on 2D range data [71]. The authors hand-construct a number of features describing the range data, allowing them to train a classifier to distinguish between rooms, corridors, doorways, and halls. Sousa et al., operating on a similar class of geometrical features, show some success applying Support Vector Machine (SVM)s to place classification, as well [91].

Shi et al. demonstrate a method for applying logistic regression to the same dataset and features as Mozos et al [88]. An interesting twist to this method is its ability to determine place categories for individual beams in a given scan, aiding in overall classification results.

In a later work, Mozos et al. use 3D range data augmented by intensity data to distinguish between types of rooms, such as offices and kitchens [70]. By constructing histogram-based features capturing local patterns in the range and intensity data, they are able to train an SVM to distinguish places classes for both full and partial panoramic scans of the environment.

Hellbach et al. propose using non-negative matrix factorization on occupancy grid data to perform feature discovery [38]. They are able to achieve high rates of classification accuracy when using this as input for generalized learning vector quantization.

Online methods are not limited to range data. In addition to distinguishing spaces based on structural features, spaces may also be distinguished by the presence of particular objects in them. Several methods examine the process of building up a semantic hierarchy of spaces based on the presence of various objects in the scene [107, 108]. For example, a cluster of chairs around a table may be learned to denote a meeting space, while a table covered with books and a computer is a work space. The presence of both near each other may further indicate presence in an office. Information such as the number and spatial relationships between a number of objects can be used to learn models of places for use in general environments. These works are largely driven by vision-based object recognition systems, whereas the focus of this work is on extracting semantic information from structure based on range data.

Much like the works above, the method introduced in this paper is intended for use as an online method. Live occupancy grid data produced by the robot may be fed into the system, which returns a semantic label. The key insight in this paper is that CNNs, commonly used in the image classification domain, can also be applied effectively to occupancy grid data, eliminating the challenges of hand-designing features.

5.2.2 Map post-processing Systems

A variety of applications focus on post-processing metrical maps to produce semantically annotated versions of the map, topological representations of the space, or both. They may employ strategies similar to online-labeling systems, or even directly use the output of such systems, but may additionally make use of spatial relationships between locations to help constrain the problem.

Beeson et al. employed an Extended Voronoi Graph (EVG) constructed from 2D occupancy grids to identify places for the purpose of topological map building [8]. In this context, places can be distinguished based on the presence of various numbers of "gateways" and "path fragments" defined by the EVG. This can also provide some general discriminative ability about the types of places. For example, an intersection can only exist when there are multiple path fragments.

Friedman et al. introduce the Voronoi Random Field (VRF) which uses a Voronoi graph to generate a Conditional Random Field (CRF), thereby incorporating spatial relationships into the classifier [27]. For example, labels will generally be locally consistent with their neighbors. Like Mozos et al., they also use AdaBoost to learn a classifier for different place types, later using the results from the AdaBoost classifier in the CRF. By taking advantage of the connectivity features, they are able to improve labeling consistency in addition to providing useful segmentations of the environment, which can be used to extract a topological representation of the space.

Mozos et al. follow up the original AdaBoost work by examining how further processing may improve the performance of the original system [72]. By applying some heuristics to correct initial labeling results, the authors are then able to segment the environment by region to produce largely accurate topological maps of several environments.

Additionally, it has been shown that Associative Markov Network (AMN)s can be useful in improving classification results in the context of map-annotation [103, 72]. AMNs take advantage of the fact that labels are spatially correlated to produce improved classification results. For example, moving 10 cm often does not change the type of location that the robot is in.

Our proposed method could be used as input to many of these systems. However, we do not

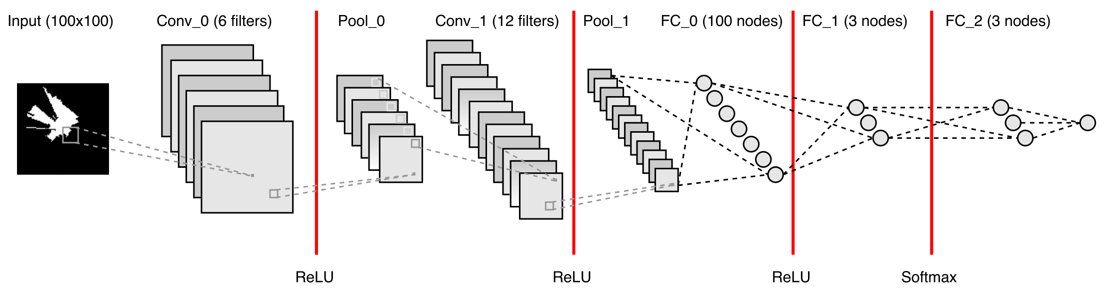


Figure 5.2: The input gridmap is fed through several convolution/pooling layers before the results are classified by a multilayer perceptron.

investigate data post-processing methods for improving our results in this paper.

5.3 Place Recognition Targeting Interactive Systems

Our target application for the semantic labels produced by our system is an interactive robot in an indoor environment. Interactions include instructing the robot to perform tasks such as delivery or finding an open meeting room, or the reverse, as when a robot describes a scene to the human. The environment may not be known in advance, so it is important that the robot can produce reasonable classifications in unknown domains from as little as a single viewpoint. For this reason, we only use live sensor data as input to the classification system (e.g. range measurements and pose). The system should also be robust to viewing angle, producing consistent labeling for the same XY location regardless of robot orientation.

The input is a fixed-size occupancy grid centered on the pose of the robot such that the robot is facing down the X-axis. As can be seen in Fig. 5.3, our occupancy grids are created by carving free space from structure, as opposed to a more traditional format in which all space is assumed free until structure is observed. Though a seemingly small difference, this distinction is critical. When marking structure, range returns falling outside the range of our fixed-size occupancy grid are lost, adding no information to the system. Space carving allows us to retain some information about distance range returns, even if our occupancy grid is small.

5.3.1 Classifying Places With CNNs

We hypothesize that the features relevant to distinguishing classes of places in occupancy grids are similar to those learned by the CNNs employed in image-based object classification tasks, therefore CNNs will also prove useful in this domain. CNNs are known to learn increasingly complex and specific features by combining results from previous convolutional layers [110]. While CNNs

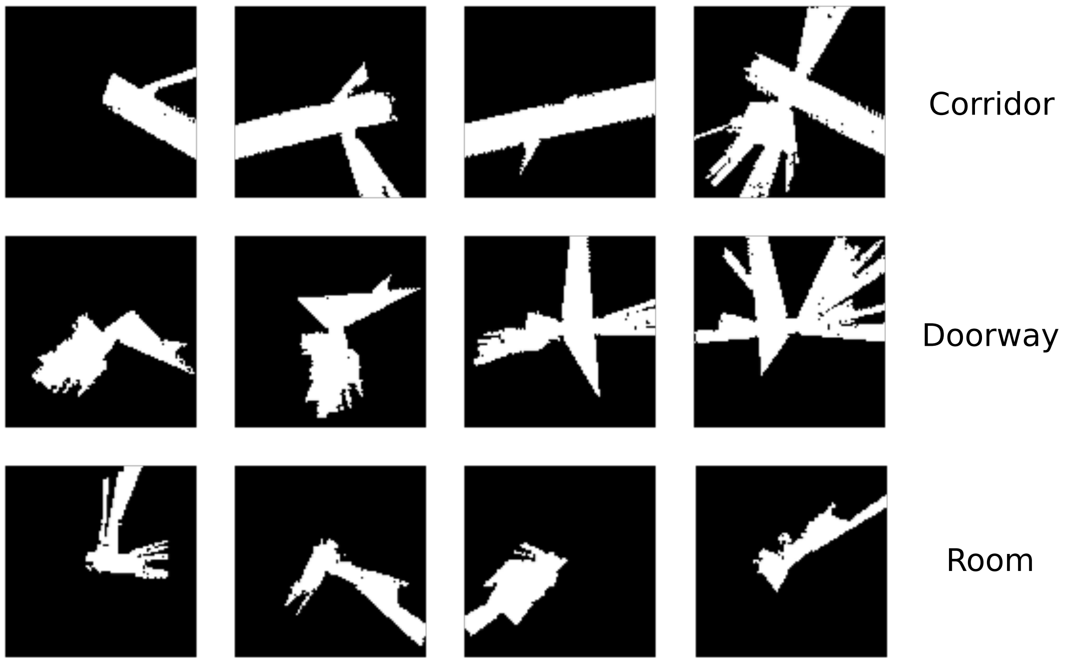


Figure 5.3: Example occupancy grids for three place classes. These serve as input to the CNN. Black denotes structure, while white denotes known free space. We assume structure by default, carving out free space based on range returns from our 2D LIDAR. This allows observations falling outside the fixed range of the occupancy grid add information to the image.

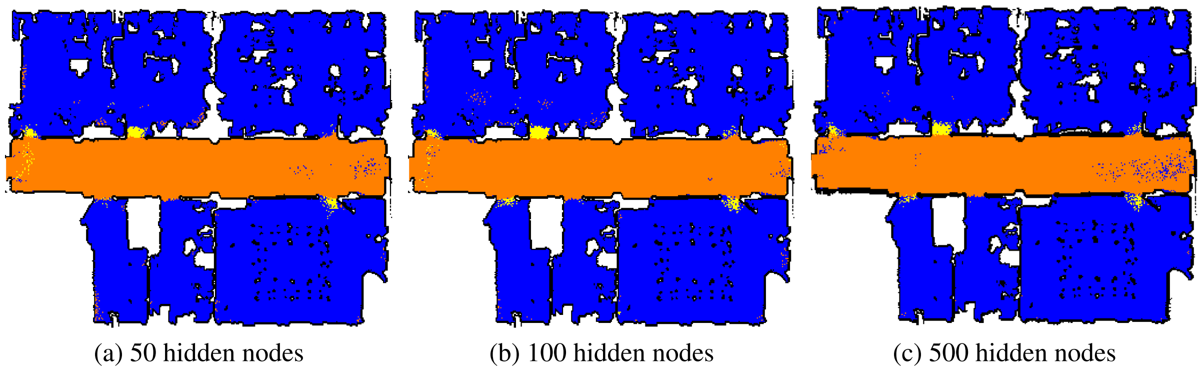


Figure 5.4: Test results on the fr79 dataset when varying the number of hidden nodes in the network. Too many hidden nodes results in overfitting the training set, making the end of the hallway challenging to classify. To few, however, and we lose the ability to classify several otherwise identifiable doorways.

containing upwards of 20 layers can be found in the literature [97], we achieved good performance with relatively small networks.

We employ a network structure loosely based on LeNet-5 [60]. Input is fed through a stacked pair of {convolution, ReLU, mean pooling layer} groupings. The feature maps are then fed into a fully connected ReLU layer before being collapsed into a probability distribution-like output across class labels by a fully connected Softmax layer. One final fully connected layer is applied to this distribution to produce a single label as the classifier's output. Our network structure can be seen in more detail in Fig. 5.2.

The final fully-connected layer after the softmax plays a critical role in classification. Using the LeNet-5 network structure, the penultimate softmax layer outputs were "scene contains a" detectors. The input occupancy grids often capture several classes in one image, especially in the case of doorways, from which both a room and a corridor may often be visible. As a result, it is not uncommon to see a situation when both the room-detecting and doorway-detecting portions of the network activate simultaneously.

This confusion often manifests in a split probability distribution, where the system is split between two or more classes. Perhaps because doorways are small compared to the other classes, this signal in tends to be overwhelmed by the nearby presence of rooms and corridors. The presence of this confusion can actually be a source of information, though. For example, the presence of simultaneous corridor, room, and doorway signals may, in fact, be indicative the doorway is the true class label, as this confusion only occurs when the robot is in a doorway. The last fully-connected layer aims to capture information based on these patterns in the final probability distribution, helping to learn an often confusing class such as doorway. Experimentally, we found that adding this final fully connected layer resulted in up to a 33% increase in correct doorway classifications.

5.3.2 Training Procedure

Neural networks are notoriously difficult to train [104]. The power to discover relevant discriminatory features based on the data also leads to a well known tendency to overfit the training set. To mitigate the problem of overfitting, we incorporate a validation set into the training procedure. At the end of each epoch, the current network is tested against the validation set. If classification performance has improved, the current network parameters are saved, and training continues. If, after a certain number of epochs, no improvement has been seen, we revert to the best parameters based on the validation set and terminate the training procedure. Final test results are then produced from a third, independent test set.

To further increase the generality of the features learned, we mitigate the impact of potential biases or limitations in the training data. For example, corridors come in a variety of widths, but if the training set only contains one example corridor of a fixed width, this information is lost. To augment the variety in training examples, we randomly apply X and Y scale transformations between 85% and 115% to training examples during runtime. Additionally, to ensure that robot orientation has a limited impact on classification results, we ensure that random orientations are selected for each training pose in the environment.

If there are widely imbalanced numbers of training examples for each class (as there are in our datasets, since doors and even corridors take up much less space than rooms), the network can be biased towards detecting the more prevalent classes. For example, doorways account for only 2.3% of labeled examples in one of our training sets.

One method for addressing this phenomenon is random reselection of training examples from the smaller pools of class examples, also known as oversampling [66]. This operation may be repeated during a training epoch until each example from the largest pool of class examples has been seen once. If we had 1000 training instances of rooms, 500 of corridors, and 100 of doorways, each room example would be presented once, each corridor twice, and each doorway ten times. In conjunction with our runtime data augmentation, this increases the number of "unique" class examples. We found that this mitigated the impact of class imbalance for our dataset. For multi-threading purposes, data may also be split into mini-batches in a similar manner.

5.4 Results

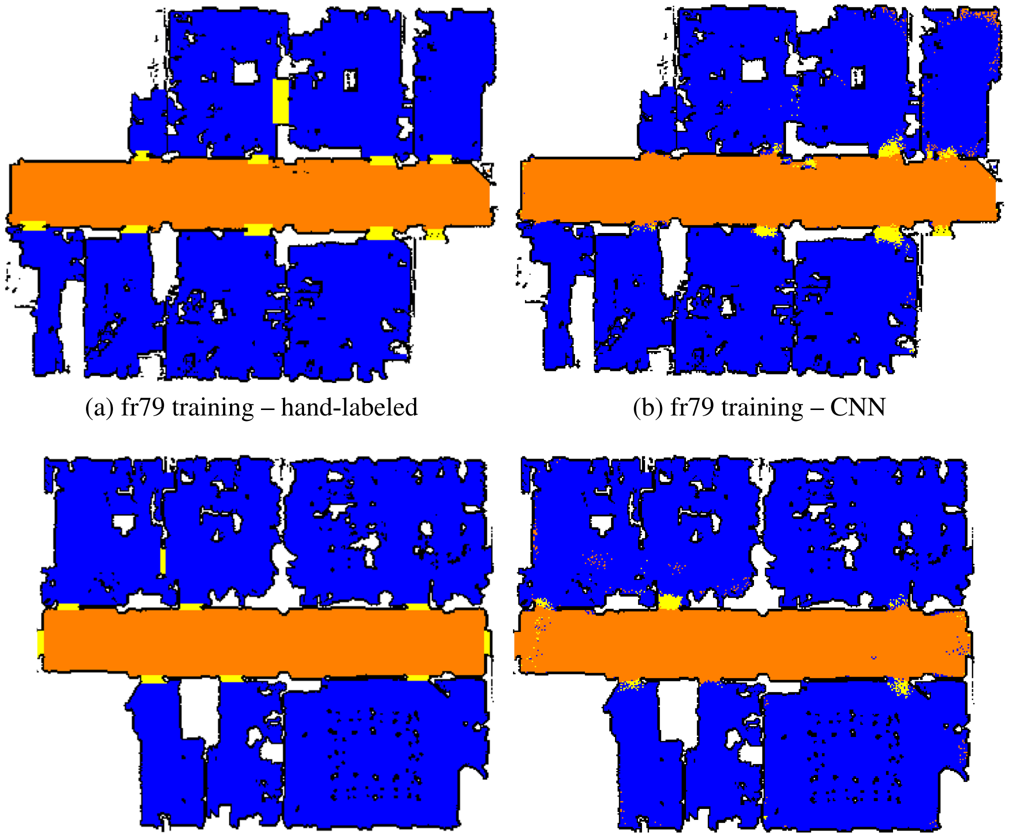
We evaluate our system on the dataset[2] used by Mozos et al. in [71]. The dataset consists of three hand-labeled buildings, each of which is split into a training half and a test half. Two, fr52 and fr79 contain examples of the classes room, corridor, and doorway. A third, fr101, contains an additional class, hall. Since we do not have another dataset containing the hall label for use as a validation set, we do not use fr101 in our tests. An example of hand-labeled training data from the fr79 dataset can be seen in Fig. 5.5a.

We use an in-house neural network library written in C to construct our CNN. Training was performed without GPU acceleration on a machine containing 32 GB of RAM and 2 2.5 GHz Intel Xeon processors, giving a total of 24 hyperthreaded cores. Training time typically took between 8 and 24 hours, depending on the structure and parameterization of the network.

5.4.1 Choosing Network Parameters

When training a CNN, the choice of the size of the input and size and number of hidden nodes can have a large impact on performance. We limited our input to 10 cm occupancy grids, which

[2]`http://www2.informatik.uni-freiburg.de/~omartine/place_data_sets.html`


(a) fr79 training – hand-labeled (b) fr79 training – CNN

(c) fr79 test – hand-labeled (d) fr79 test – CNN

Figure 5.5: Hand-labeled input for the fr79 dataset (a) (c) compared to labels produced by our CNN classifier (b) (d). The data seen in (a) was used to train the CNN, while training data from fr52 (5.6a) was used as a validation set to prevent overfitting.

balance capturing large areas while preserving fine-scale structure needed for doorways.

Experiments were performed for images capturing 4×4 m through 10×10 m regions surrounding the robot. We found that selecting smaller regions often improved performance on doorways, but at the cost of distinguishing classes such as corridors and rooms. This result can intuitively be linked back to the scale of the places in question. Doorways occupy small areas, thus small occupancy grids filter out distracting information. Conversely, many parts of a corridor or room may only be adequately captured at a larger scale. As a result, we opted to focus on the largest, 10×10 m occupancy grids, as they seemed best suited for capturing relevant information about all classes.

Based on this input image size, convolutional filter sizes were set to 9×9 pixels. Additional filter sizes were not explored, as these settings seemed to work well. Six first-level features proved sufficient to capture low-level details in the occupancy grids. Variation in the number of second-level filters had only marginal impact on the results. Instead, we focus our analysis on the number

	Room	Corridor	Doorway
Room	98.3%	1.1%	0.6%
Corridor	0.7%	99.1%	0.2%
Doorway	24.1%	60.4%	15.5%
Overall Accuracy			97.8%

Table 5.1: Classification confusion results corresponding to fr79 test (d), with true values separated by row. Our method performs well on room and corridor classes, but struggles with doorways.

	Room	Corridor	Doorway
Room	97.8%	2.1%	0.1%
Corridor	7.4%	92.3%	0.4%
Doorway	51.7%	32.4%	16.9%
Overall Accuracy			95.3%

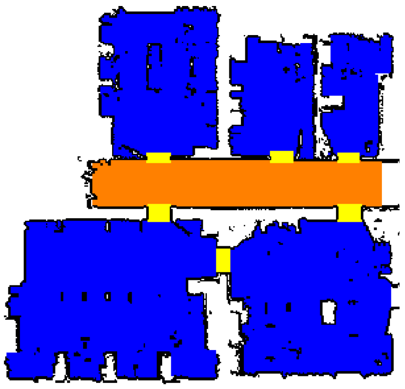
Table 5.2: Classification confusion results corresponding to fr52 test (d), with true values separated by row. Accuracy on corridors decreases to that seen on fr79 test. This can likely be attributed to overfitting the model to the limited training data. In particular, note the difference in structure between the examples of dead ends in the training and test sets. Corridor errors are overwhelmingly focused in these dead ends.

of fully connected nodes following the convolutional stages of the network. If the fully connected layer is too large it becomes easy for the network to overfit the training data. However, if the fully connected layer is too small, the network will be insufficiently powerful to distinguish between classes.

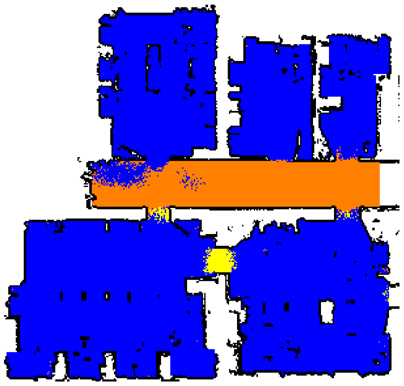
We present results from three networks trained on the fr79 dataset, with a layer of 50, 100, and 500 fully connected nodes after the final pooling layer. Test results in the form of annotated maps can be seen in Fig. 5.4. Accuracy on the test set varied by less than 1% between the three network structures, but even so, qualitative differences in how noise manifests can be seen when comparing the annotated results. With 500 hidden nodes, signs of overfitting appear, particularly in the corridor classifications. With only 50 hidden nodes, doorway classification results begin to deteriorate. As a result, the rest of our tests are performed with a network containing a layer of 100 hidden nodes following the final pooling layer.

5.4.2 Map Annotation

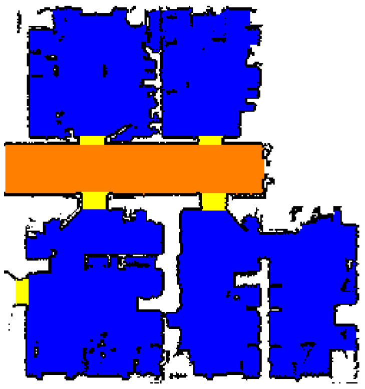
Though our system is designed for use online, it is easy to demonstrate its performance across viewpoints in an environment by using it in a map annotation task. We train two networks, one using the fr79 training set, validated against fr52 training set during training, and the other trained



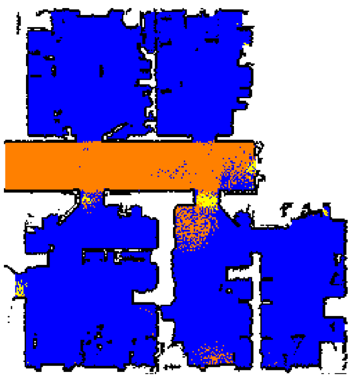
(a) fr52 training – hand-labeled



(b) fr52 training – CNN



(c) fr52 test – hand-labeled



(d) fr52 test – CNN

Figure 5.6: Hand-labeled input for the fr52 dataset (a) (c) compared to labels produced by our CNN classifier (b) (d). The data seen in (a) was used to train the CNN, while training data from fr79 (5.5a) was used as a validation set to prevent overfitting.

by reversing the sets. Our network structure can be seen in Fig. 5.2. Then, both networks are evaluated against the fr52 and fr79 test sets.

Classification results trained and tested on fr79 can be seen in Fig. 5.5, while results trained as tested on fr52 can be seen in Fig. 5.6. Hand-labeled training sets were used for training and validation, while test sets were reserved for evaluation. Results generated by the CNN classifier are visualized for both the test and the training sets.

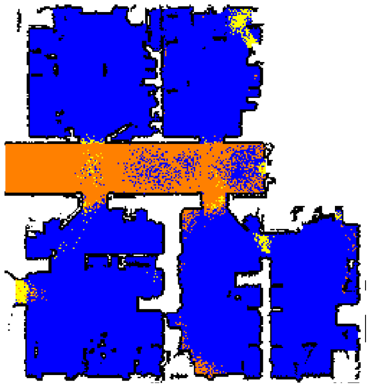
Accuracy on both test sets are comparable to or better than those seen in the literature, achieving 97.8% accuracy on fr79 and 95.3% accuracy on fr52 (see Table 5.1 and Table 5.2 for class specific breakdowns). In comparison, Mozos et al. report 93.4% accuracy on fr79 nad 92.1% on fr52 with their sequential AdaBoost method [71]. Labels show large degrees of spatial consistency, in particular for rooms and corridors, although there is some speckled noise. It is likely that this noise could be effectively filtered in postprocessing steps.

As is the case in the literature, doorways are challenging to effectively classify. As documented in Sec. 5, this can be attributed to the fact that views from doorways inherently contain other classes, as well as the small amount of training data generated by a class occupying such a small area. On the fr79 dataset, we are only able to achieve 15.5% accuracy on doorways in the test set. Mozos et al. only report training error for individual classes, but they, too, report the highest individual class error was seen for doorways. While disappointing, further investigation of the failures shows that a large portion of doorway detections register just beyond the bounds of the labeled regions. Given the subjective nature of the doorway annotations in the training and test data, these results could still be quite serviceable in practical applications. A larger number of training examples would also likely facilitate learning.

5.4.3 Performance on New Environments

In this section, we examine how the classifiers trained in Sec. 5.4.2 perform in new environments from which no training data has been seen. We would like to know if the internal representations of classes being learned are general enough to be used in novel domains. To this end, we evaluate the fr79 classifier on the fr52 test set and the fr52 classifier on the fr79 test set. Annotated map data for both tests can be seen in Fig. 5.7.

Overall accuracy on both test sets remains high, but does decrease compared to when a classifier trained on the other half of the same building is used. This is likely due to overfitting building specific characteristics. For example, the end of the corridor in the fr52 training data is very different from that in the fr79 test data. As a result, this region of fr79 is mostly misclassified as "room." Likewise, the doorways in both training environments are largely dissimilar from those in the test environments, making them even more challenging to learn.


(a) fr79 CNN on fr52 test

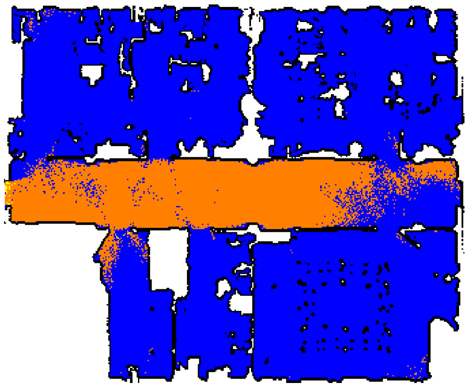

(b) ft52 CNN on fr79 test

Figure 5.7: Test results for fr52 and fr79 based on training data from the other building. Thought overall accuracy still exceeds 90%, performance decreases, likely due to overfitting building specific characteristics.

	Room	Corridor	Doorway
Room	97.0%	1.8%	1.2%
Corridor	16.6%	81.2%	2.2%
Doorway	29.7%	47.3%	23.0%
Overall Accuracy			93.2%

Table 5.3: Percentage of proposed labels (column) compared to the true label (row) for fr52 test data, trained on fr79.

	Room	Corridor	Doorway
Room	98.8%	1.1%	0.0%
Corridor	24.4%	75.6%	0.0%
Doorway	73.9%	20.8%	5.3%
Overall Accuracy			92.4%

Table 5.4: Percentage of proposed labels (column) compared to the true label (row) for fr79 test data, trained on fr52.

These results, while promising, do highlight the challenges of training robust CNNs. Though they may learn powerful representations of the data, they often require vast amounts of data to prevent overfitting. It is likely that the datasets employed in this paper are insufficiently large to avoid these issues, even after performing scale and rotation transformations to extend usefulness of the data.

5.5 Conclusion

In this paper, we have demonstrated that CNNs can be used to great effect in learning semantic place labels from 2D range data commonly collected by robots. Results generalize well between environments, but could be improved by increased variety in the training data. Our method performs as well or better than existing work.

Our learned semantic place classifier is well-suited for use in DART-style navigation. Transitions between different classes of places or the detection of a specific type of place can be used as switching conditions to change navigation control policies. Cues based on structure are particularly interesting, the structure of a place is often related to the types of actions that may be taken there. Our classifier is cheap to run and operates on easily simulated sensor data, making it ideal for evaluating the large number of simulated trials required by DART.

CHAPTER 6

Learning Control Policies

The selection of control policies made available to a robot are critical in determining the types of environments in which a robot can operate and how it moves about them. In Chapters 3 and 4 we introduced DART and discussed the advantages of navigation based on chaining together closed-loop control policies. In our initial experiments, these control policies were hard-engineered. While policies could be hand-engineered for some situations, this strategy scales poorly and requires expert domain knowledge. It is also challenging for experts to anticipate and evaluate how the robot will react in many configurations of the environment (e.g. a building made largely of glass or in a house with non-standard sized doorways), leading to potential brittleness in the resulting algorithms. By learning policies based on real data and experiences, we aim to capture important aspects about the task at hand and enable policies to transfer well to previously unseen scenarios.

In this chapter, we propose a method for learning policies from demonstrations. Learning policies from demonstration provides several benefits over alternative learning strategies. For example, one alternative is unsupervised learning (e.g. reinforcement learning [48]), but this typically requires an expert to define a reward function; this is frequently impractical, as the goals of many tasks are difficult capture accurately. Even with a reasonable reward function, the robot must be given a safe environment in which to learn, or it may it damage itself in the process.

Demonstrations, on the other hand, are comparatively easy to provide, even for non-expert instructors [5]. This provides regular users the important capability to teach robots new skills based on their needs or to help robots adapt their existing behaviors to challenges specific to a particular environment. The supervisor can ensure the safety of demonstrated actions actions. The key challenge in learning from demonstration is collecting enough demonstrations to accurately capture the desired behavior.

Our method produces a similarity-based scoring function that evaluates the utility of the robot being in particular navigation states. For example, a function trained from demonstrations of wall following behavior will return higher scores for poses oriented parallel to nearby wall-like structure. A map showing high-utility states can be seen in Fig. 6.1. We show that this function is

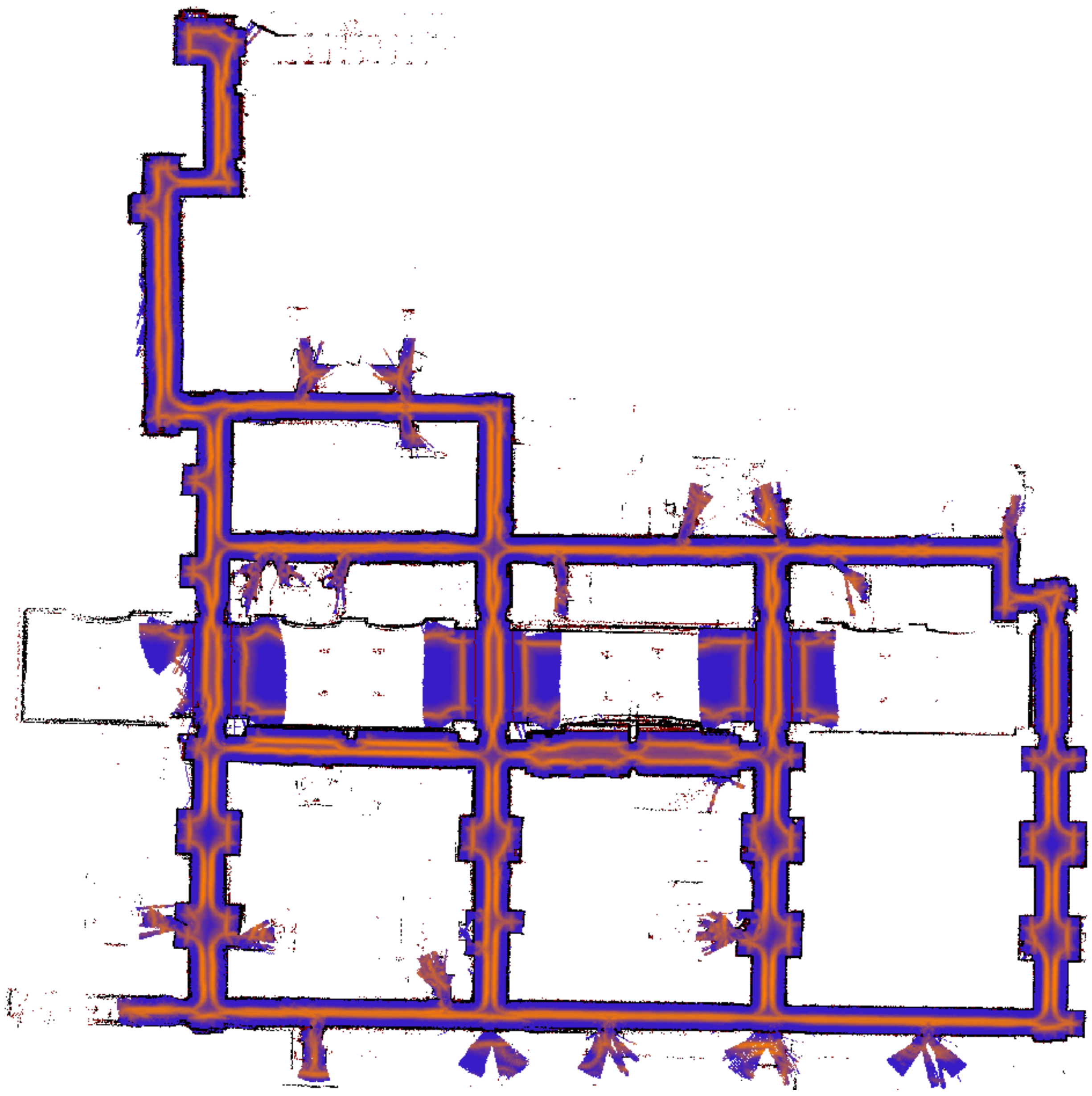


Figure 6.1: A map showing the reward gained for visiting various states for a learned left wall follower, with the highest reward states in orange and the lowest in blue. The values were generated by exhaustively sampling viewpoints at from free space in the map at 5 cm translational increments and 1 degree orientation increments and selecting the maximum reward for that location. The bright bands that trace paths near the walls show that the system has learned a wall-following policy.

straightforward to integrate into a traditional potential-field based navigation framework, allowing us to compose functions rewarding high level behavior with lower level functions rewarding behaviors such as obstacle avoidance. The function learning algorithm involves only a few, easily-tunable parameters. We present results for two learned behaviors suitable for many indoor navigation tasks: left and right wall following. Wall following policies are powerful navigation strategies for several reasons. First, the robot does not require accurate global localization behavior to execute the policy, making it robust to failures in localization subsystems. Second, walls are easily detected with a variety sensors commonly available to robotic systems such as LIDAR. Finally, wall following behaviors are well suited to indoor environments composed largely of hallways, as they can be used to navigate to virtually any area. By switching the wall to follow at strategic points, a wall following robot can execute a wide variety of routes. The main contributions of this chapter are:

- A method for constructing scoring functions that are well suited to navigation control policies.
- A strategy for integrating this scoring function into a potential-field based navigation framework.
- Evaluation of learned scoring functions in noisy simulations based on several real-world datasets.

6.1 Related Work

Though it is sometimes feasible for experts to manually program robots with the necessary logic to perform tasks autonomously, this solution does not scale well or guarantee working solutions. Hand-crafted systems may fail to anticipate challenging corner cases, and even when they do, are time consuming to construct. Instead, it is preferable work within learning frameworks that can be re-used in different applications. It is particularly desirable for robots to be able to adapt their behavior on-the-fly, learning new behaviors based on shortcomings encountered over the course of real-world task execution. In the following section, we discuss examples of learning techniques that have been applied to robotic motion problems.

6.1.1 Reinforcement Learning

Reinforcement Learning (RL) is an unsupervised leaning process, meaning the system is allowed to learn without expert input, instead strategically exploring the action space to discover optimal behavior(s). The learning agent is provided with a reward function that indicates the utility of taking particular actions at particular states. Over time, the learned policy will improve, allowing the agent to collect higher rewards. If the reward function is well designed, this should yield a policy that achieves the accomplishes the desired task.

In an early example of RL in robotics, Mahadevan and Connell teach a robot equipped with sonar sensors to push boxes around an environment [65]. They employ a similarity-based clustering technique to reduce the otherwise intractable state space, using both a hand-coded similarity metric based on weighted Hamming distance as well as another metric based on the statistical properties of the clusters to group similar states. They show that they are able to reduce the size of the state space sufficiently to learn policies that are comparable to a hand-coded baseline within twenty fixed-length learning trials using either metric.

Bagnell and Schneider describe a technique for learning a flight controller for an autonomous helicopter through policy search techniques [6]. In domains such as flight control, there exist dangerous regions of the state space that should be avoided, lest the robot crash and suffer damage. The authors artificially limit the space of possible controllers to impose restrictions, such as ones motivated by safety, on the learning process. This also enables a reduction of otherwise large and intractable state spaces, making learning viable.

Reinforcement learning has also been applied to the domain of robot soccer, such as when Stone et al. showed that simulated RoboCup agents could learn to play keepaway [94]. To make the problem more tractable, the action space consisted of several pre-determined macro-actions such as $pass\_to(agent)$ and $hold\_ball()$. The agents are able to learn effective strategies for retaining possession of the ball for varying sizes of play area and numbers of players.

While these examples show the viability of RL is some robotic tasks, in general, it is challenging to apply RL to these domains for several reasons. First, reward functions are often difficult to construct, even for experts. Well-designed reward functions are critical for ensuring robots learn the desired behaviors. Second, real-world experience is expensive to collect and may require extensive setup to ensure a safe learning environment. This can make it impossible to collect the data necessary for the robot to learn. Finally, robotic tasks typically have large state spaces which present particular problems to RL techniques. These spaces can be prohibitively expensive to explore, preventing RL methods from converging efficiently on good policies. These difficulties cause many in robotics to turn towards supervised learning techniques.

6.1.2 Learning from Demonstration

Learning from Demonstration (LfD) is a supervised learning technique in which an instructor shows a robot, frequently through direct manipulation of the robot, how a task is performed. Demonstrated action sequences captured in a variety of ways, from direct teleoperation to passive observation and imitation of an expert demonstrator. In the former case, the learning system may try to learn a direct mapping between the observed states and the actions executed by the robot, while an imitation system may try to classify the actions of the demonstrator in terms of discrete actions available to the robot.

Pomerleau provides one of the earliest examples of the application of LfD to autonomous vehicles [80]. In this work, a neural network is trained to keep an autonomous vehicle centered in the road. Front-facing images of the scene are used as input, which the network learns to map to set of discretized steering wheel angles. Demonstration data consists of sequences of human-generated steering wheel positions and corresponding images of the road in front of the vehicle at the time the wheel positions were observed. However, the author encounters a problem common in LfD domains: demonstrations often do not cover a sufficient portion of the state space to learn robust behaviors. In this case, the vehicle is unable to recenter itself in the lane once it drifts out of lane center, since the example behaviors do not encounter such a state. To learn recovery behaviors, the training data is augmented with synthetic samples generated by shifting the image data and steering inputs laterally.

LfD has also made appearances in the domain of robot soccer. Browning et al. show that human demonstrations of soccer actions (such as "kicking") can be generalized into policies for use by the robot [11]. Humans provide demonstrations of discrete actions in the form of teleoperated examples. The authors employ Locally Weighted Regression (LWR) to map directly from observed states to motor actions. The intuition is that the robot should take similar actions for similar sensor states. LWR works well with sparse data, interpolating between the samples to fill out the demonstrated portion of the state space. This strategy works well in the authors' domain, where relatively few demonstrations provide effective bounds to the useful portion of the state space. However, LWR can be ill-behaved in cases where actions must be extrapolated into unexplored regions of the space.

Calinon and Billard use Gaussian Mixture Regression to teach a robot referee basketball hand signals [13]. First, the desired motions are demonstrated either by an instructor instrumented with motion sensors or by manipulating the robot directly. As demonstrations are gathered, mixtures of Gaussians are fit to the set of trajectories. The learned trajectory is extracted by estimating the maximum likelihood trajectory from the learned mixtures. This strategy is well suited for this domain, as gestures will always follow a fixed sequence of motions. However, this form of memorization and repetition would not scale to a domain such as ours, where we wish to generate appropriate actions in environments the robot has never previously observed.

Akgun et al. demonstrate a technique for learning based on sparsified, "keyframe" demonstrations [2]. Keyframes are extracted at important points in the demonstrations, providing compact summaries of the trajectories. Generalizations of the demonstrations are extracted by clustering similar keyframes. These clusters form the basis of pose distributions which can be visited in sequence to reproduce the desired behaviors. The authors demonstrate their technique on two robot motions: scooping and pouring of coffee beans.

The described methods are well-suited to learning single action policies, but traditional LfD techniques struggle with policies composed of multiple sub-policies. For example, the policy-switching behavior encoded in a Finite State Machine (FSM) is difficult to represent, as the same observation maps to multiple possible actions. Grollman and Jenkins propose employing infinite mixtures of experts to address this limitation [32]. The demonstration data are partitioned into subtasks with associated "experts" on the basis of how well an expert can explain the action taken. If no expert is suitable, a new expert is created and added to the pool. This representation allows for the possibility of experts overlapping in observation space, allowing them to learn FSM-style policies.

Niekum et al. propose learning FSM representations directly, demonstrating their technique on a robot furniture assembly task [73]. The authors first segment the demonstrations into categories based on the distinctness of motions, providing an initial set of states and transitions. This baseline FSM is further optimized by semantically splitting states based on differences between the current FSM and the actions seen in demonstrations.

Poor performance is common when LfD systems attempt to extrapolate behaviors to undemonstrated portions of the state space. Demonstrations tend to be focused on very specific, "good" sections of the space, but as a result, learned policies are ill constrained in the unexplored regions [5]. Given the large state spaces involved in robotic domains, ensuring that demonstrations sufficiently cover the state space is a recurring problem.

6.1.3 Apprenticeship Learning via Inverse Reinforcement Learning

The challenges of RL and LfD have led to hybrid approaches that try to leverage the strengths of both. A simple strategy is to use LfD to initialize policies that are then optimized with reinforcement learning. Kormushev et al. use this strategy to teach a robot the dynamically complex task of flipping a pancake [50]. By initializing their RL system with a policy based on a single successful demonstration, the authors are able to reduce the number of expensive, real-world trials necessary for convergence on a good policy.

Apprenticeship learning has emerged as another successful strategy merging the strengths of both RL and LfD. The key idea behind apprenticeship learning is that robust policies can be challenging to learn from demonstrations, but it is often easier to learn good reward functions from them. The process of learning this reward function from demonstrations is often referred to as Inverse Reinforcement Learning (IRL).

One of the earliest examples of apprenticeship learning comes from Abbeel et al., who demonstrated a method for learning controllers for executing challenging aerobatic maneuvers on an autonomous helicopter [1]. Rather than trying to learn the control outputs directly, the authors instead observe an expert pilot executing challenging acrobatic maneuvers and use these demonstrations to construct a function rewarding motions similar to those taken by the expert. This function is used to model the likely trajectory the expert was trying to follow, subject to a model of helicopter control. Then, the maneuvers may performed autonomously by trying to reproduce the learned trajectories.

Kim and Pineau leverage IRL in the domain of robotic wheelchairs, showing that it can be used to help their robot drive in more socially acceptable ways [47]. Driving “politely” in crowds while still efficiently maneuvering towards one’s goal is a challenging task for a robot. Rather than manually constructing a reward function aiming to capture these mutual goals, the authors learn a reward function based on demonstrations of humans teleoperating the robot in social settings. The learned reward function is integrated directly into a short-horizon path planning system as an estimate of path utility. The resulting trajectories demonstrate more “human-like“ behavior than traditional, shortest-path methods, indicating that the learned reward function captured important aspects of the task.

The problem solved by Kim and Pineau most closely resembles our own, and their results suggest that similar learning strategies could work well for learning closed loop control policies suitable for DART. The differentiating factor is the level of complexity of the desired policies. At its core, the policy learned by Kim and Pineau is a version of standard goal pursuit with obstacle avoidance. The policies we wish to capture must leverage spatial and semantic knowledge about the robot’s surroundings to take appropriate actions, using the robot’s current observations as a cue of where to drive rather than an explicitly defined goal.

In the remainder of this chapter, we describe our method for constructing appropriate features for our problem space and using them to train a scoring function from teleoperated demonstrations. The learned function rewards similarity between new states and model of states encountered during the demonstrations. This similarity function is intentionally designed to be composited with other scoring functions such as penalties for going too close to obstacles, allowing us to impose additional safety constraints on the final behavior.

6.2 Learning Navigation Policy

In this section, we will discuss a method for learning control policies from demonstrations of navigation tasks. In particular, we focus on a class of navigation tasks without explicitly specified end states, which instead can be described as behaviors or closed-loop control policies.

These capabilities are interesting for several reasons. First, they can be used to build sequences of macro-actions, compactly describing large routes through the environment. In conjunction with a suitable set of policy-switching conditions such as the one introduced in Chapter 5, these form the core building blocks of DART. Second, many of these policies have straightforward linguistic descriptions, providing opportunities to build human-friendly interfaces for giving or receiving navigation instructions. Finally, these policies can also be executed without pre-existing map knowledge, enabling robots to execute routes in previously unseen environments.

Our goal is for a robot to learn to drive in a manner similar to human demonstrations of the desired policy. As previous work has shown [80], it can be challenging to learn a policy directly from the demonstrated data, since many regions of the state space are not adequately covered. Learning a function that rewards good behavior is often easier.

Intuitively, a "good" policy will prefer to visit states with key features similar to those seen in demonstrations of the optimal behavior. For example, a wall following policy will likely learn to prefer states in which there is a continuous block of vertical structure on the appropriate side of the robot. Also, since we know that our robots move continuously through space, we know that good states will typically be adjacent to other good states. This suggests that a robot could reproduce a demonstrated behavior by pursuing a greedy policy of moving towards the adjacent states states which are most similar to those encountered in the demonstrations.

6.2.1 Riding the Behavior Luge

A class of robot motion policies already exist that pursue this strategy: potential field navigation methods. Potential field methods are a traditional style of navigation and control used in some robot systems. A robot can compose several potential functions, such as a repulsive function which produces higher values as the robot approaches obstacles and an attractive function which produces lower values as the robot approaches its goal, and use the gradient of the resulting cost surface as a control input. Each potential function captures an important aspect of the task (e.g. don't hit things, move towards this location), enabling high level behavior to emerge from the combination of more easily specified individual tasks.

The potential function tends to decrease towards the goal, resembling a "luge" chute. This has the benefit of allowing the environment to dictate the level of precision necessary to safely control the robot. As the luge tightens, the robot will try to follow the center of the channel more precisely, since the cost of deviation is higher. As the luge widens, however, the robot can be lazier in its control.

The simplicity of potential methods comes with a trade off; they are prone to becoming caught in local minima since they only greedily consider the next action to take. This has lead to their popularity dwindling in recent years. However, DART presents an opportunity to revisit these methods. While a lack of global planning guarantees is problematic for a traditional navigation framework, DART can take the weaknesses of a particular control policy into account, only using it in portions of the environment where it is appropriate.

We propose that safe policies exhibiting high level behaviors other than goal pursuit can be implemented in a potential-field framework by composing behavior-derived gradients with traditional repulsive ones. In the following section, we describe a technique for learning a reward function capable of generating "behavior luges" such as the ones see in Fig. 6.1. We follow a standard formulation for machine learning, constructing our reward function in terms of a set of features describing the state space and a set of weights which emphasize features important to a particular policy. We construct a controller to follow these channels, allowing the robot to reproduce the desired behavior. In scenarios where the channels guide the robot into unsafe states, a repulsive field added on top of the behavior ensures that the robot avoids harm.

6.2.2 Feature Space

Choosing the appropriate features is an important aspect of any learning from demonstration task. Desirable features will typically provide a compact description of the robot's observations, mitigating issues arising from high dimensionality such as data sparsity. We denote the robot's observations from its current state as $\boldsymbol{z}$. In this section, our goal is to select a function $f(\boldsymbol{z}) \to \boldsymbol{x}$ which maps observations to a feature vector $\boldsymbol{x}$.

In this work, we will focus on features that can be derived from the range measurements collected by common robot sensors such as LIDAR, RGB-D sensors, and sonar. These sensors are capable of capturing important structural information about the environment and are commonly used in navigation tasks. These types of measurements are well-suited for use in DART, as they are easy to synthesize based on robot-built maps of the environment.

Our features must capture several critical characteristics of the environment around the robot to support behaviors such as wall following. The robot must be able to distinguish walls to follow along, indicating that we must have at least some ability to identify vertical structure. There may be more than one wall to follow, so the robot must also understand the spatial relationship between it and candidate walls, so that the correct wall may be selected. Proximity to the wall is also important for distinguishing between potential walls to follow, as well as for establishing a reasonable distance to maintain between the robot and the wall.

The robots employed in this work are equipped with 3D LIDAR capable of generating hundreds of thousands of range measurements per second. These raw measurements possess many of the desirable characteristics for features to work with but require some additional processing. We

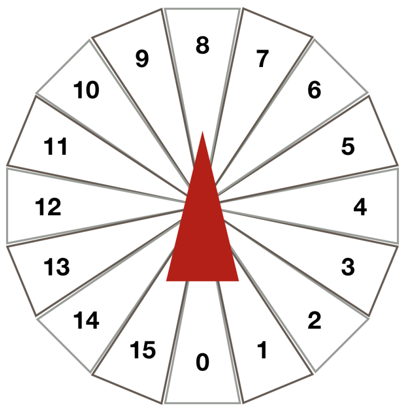


Figure 6.2: Features are constructed from range measurements falling within specific regions (*segments*) around the robot (red triangle). The feature vector is constructed by proceeding by extracting features from each segment in counter-clockwise order, beginning at the back of the robot. This ensures that features have a consistent relationship with the robot's current pose in the world.

employ the vertical structure extraction techniques described in our previous work [30] to produce a 2D compression of the 3D data. This provides an efficient filter for identifying the location of wall-like structure relative to the robot.

This filtered data is used to generate summary statistics about the location of interesting structure around the robot. To compute these statistics, we first split the 360 degree view around the robot into equally sized wedges which we call *segments*, as seen in Fig. 6.2. We assign observations of structure to segments based on their relative location to the robot. Once all observations have been assigned to a segment, we compute statistics on a per-bin basis. In this work, we compute the mean distance to structure relative to the robot in a total of 16 segments as use the means directly as our feature vector.

More segments and statistics allow the feature space to represent the state of the world in greater detail, but increasing the number of features also increases the number of training samples necessary to sufficiently cover the feature space. We experimented with several settings for this parameter and found that, for fewer that 16 segments, the robot loses its ability to discern hallway-sized openings in the environment, preventing it from learning the desired policy. Including more segments did not noticeably impact the resulting learned behaviors, leading us to opt for the more compact representation.

Similarly, we experimented with a pair of potential segment statistics. In addition to evaluating the mean distance to structure within a segment, we also considered the minimum range to structure. We found that both statistics can be used to learn wall following behaviors, but that the reward functions learned with minimum range features resulted in large discontinuities between spatially close viewpoints compared to functions learned with mean range features. This is because the minimum range is more easily affected by a single outlier measurement than the mean, which acts to smooth the transition of extreme observations between segments.

6.2.3 Learning a Reward Function

In this section, we describe a formulation of a reward function suitable for generating behavior potential fields and propose a strategy for learning the parameters of that function from demonstrations. We provide the robot with demonstrations $d \in \mathcal{D}$ showing examples of desirable behavior. Each demonstration consists of a sequence of n observation-action pairs:

$$d_i = \{(z_{i1}, a_{i1}), \ldots, (z_{in}, a_{in})\} \tag{6.1}$$

As introduced in the previous section, we map these observations to a feature vector $\boldsymbol{x}$ describing the robot's current state in the environment. For convenience, we will refer to this descriptor as the "state" for the remainder of this work. Our goal is to construct a reward function $R(\boldsymbol{x})$ which gives high rewards for being in states with similar to those seen in demonstrations (and conversely low rewards to dissimilar states), encouraging the robot to visit such states in the future.

A naïve approach might use a nearest neighbor strategy, using inverse Euclidean distance as a reward function. This has the effect of rewarding states that are close to previously-seen states, while penalizing states that differ significantly. Though simple to implement, this strategy has obvious flaws for use as the basis of a potential function. First, the cost to compute the reward increases linearly with the number of demonstrations, placing practical limitations on the amount of training data that can be used. Second, nearest neighbor strategies also suffer in high dimensional spaces, frequently running into problems with data sparsity. As dimensionality grows, more samples will be needed to discern relevant patterns in the data.

An alternative approach is to construct the reward function based on the likelihood of observing a given state. Let the states observed during optimal task execution follow some distribution $\mathcal{P}$ with parameters $\boldsymbol{\theta}$. The probability density function of the distribution can be given as $f(\boldsymbol{x}|\boldsymbol{\theta})$. A possible reward function, then, is:

$$R(\boldsymbol{x}) = f(\boldsymbol{x}|\boldsymbol{\theta}) \tag{6.2}$$

We hypothesize that "good" states will tend to share similar qualities, resulting in clusters of such states in particular regions of the feature space. This clustering phenomenon will be reflected in $\mathcal{P}$, which will have higher density (and thus higher reward) near these clusters.

The challenge lies in the fact that distribution $\mathcal{P}$ is not known. Instead, we must estimate the distribution from the demonstration data. Thus, our goal is to determine parameters $\boldsymbol{\theta}$ which maximize the likelihood of observing the states seen in the demonstrations.

While it is preferable to model the joint distribution of the features directly, this approach has practical limitations. As the number of features N becomes large, it becomes costly to collect sufficient sample to accurately fit the distribution. This can lead to overfitting to the demonstrations, degrading performance in new environments.

To address these concerns, we propose a simplifying assumption: that the features in $\boldsymbol{x}$ are independent. This allows us to model the distribution $\mathcal{D}_i$ of each feature $x_i \in \boldsymbol{x}$ separately. As a result, the reward function given in Eq. (6.2) can be updated to:

$$R(\boldsymbol{x}) = \prod_{i=1}^{N} f(x_i|\theta_i) \tag{6.3}$$

where θ_i are the parameters of $\mathcal{D}_i$.

In reality, features are likely to be correlated. However, instead of fitting a full joint distribution (which, due to its large size, would be prone to over-fitting) we fit a model that assumes independence between variables. This reduces the complexity of the distributions we are trying to learn. Instead of learning one N-dimensional distribution, where the number of parameters may grow exponentially with N, we instead can learn N one-dimensional distributions, reducing the impact of dimensionality on data collection. We also introduce some robustness to overfitting: patterns in a single feature distribution are more likely to generalize well to a new environment than complex patterns between several features. The net result is a reward which transfers well to new environments and can be learned from fewer demonstrations.

6.2.4 Choosing and Fitting a Distribution

Our reward formulation intentionally leaves the choice of distribution open. However, we suggest that Gaussian Mixture Model (GMM)s have several qualities that make them well suited for our application. GMMs represent probability distributions as a weighted sum of two or more Gaussian distributions. GMMs are capable of representing multi-modal distributions, allowing them to accurate capture a wide variety of phenomena. The probability density function of a Gaussian mixture model varies smoothly, producing smooth, well-behaved gradients useful in later optimization and control. Gaussian mixture models are also compact and easy to fit with standard algorithms.

We fit Gaussian mixtures to our training data using a standard algorithm described by Bishop [10]. First, we use the K-means algorithm to cluster our training data to construct an initial estimate of the maximum likelihood Gaussian mixture model. We then use the EM algorithm refine this estimate, maximizing the likelihood of observing the demonstration states given the parameters. Given this choice of distribution, the reward function can be expressed as:

$$\begin{aligned} R(\boldsymbol{x}) &= \prod_{i=1}^{N} \sum_{k=1}^{K} \phi_k f(x_i | \theta_i^k) \\ &= \prod_{i=1}^{N} \sum_{k=1}^{K} \phi_k f(x_i | \mu_i^k, \sigma_i^k) \end{aligned} \tag{6.4}$$

where μ_i^k and σ_i^k are the mean and variance of the k^{th} mixture describing the i^{th} feature, and ϕ_k a mixing coefficient.

In practice, the values of many features occur with low probability, leading to near-zero values in the density function. This can lead to problems with numerical precision when representing rewards. To combat this problem, we employ the standard practice of working with the log-probability, leading to the modified reward function:

$$R(\boldsymbol{x}) = \sum_{i=1}^{N} \ln \big(\sum_{k=1}^{K} \phi_j f(x_i | \mu_i^k, \sigma_i^k) \big) \tag{6.5}$$

6.2.5 Constructing the Behavior Gradient

In this section, we describe how to incorporate the learned reward function into a potential style control framework. Normally, a gradient draws the robot towards this goal, rewarding the robot for moving in that direction. The closed-loop behaviors we are interested in lack such a goal, necessitating some other motion-incentivizing force. We show that the reward function can be used to calculate two separate gradients: a corrective gradient providing translational corrections to the robot's distance (e.g. moving it closer or further from the wall) and an orientation gradient which rewards forward motion in the correct direction.

The reward function given in Eq. 6.5 varies as the robot translates and rotates in space. We construct the described gradients in two steps. First, we determine the optimal orientation for the robot to travel in, resulting in the orientation gradient. Then we fix the orientation and evaluate the corrective translational gradients.

To evaluate possible orientations, we synthesize and evaluate feature vectors for viewpoints at fixed intervals in orientation space based on the robot's current range observations. In our implementation, we synthesize viewpoints at 1 degree increments in the range of ± 90 degrees from the robot's current orientation. This sampling range ensures that the robot will consider any orientation which would move the robot forward from its current state. To determine the optimal orientation, we simply select the orientation which achieves the maximum reward. From this, we construct an orientation gradient indicating the direction the robot should travel. The orientation

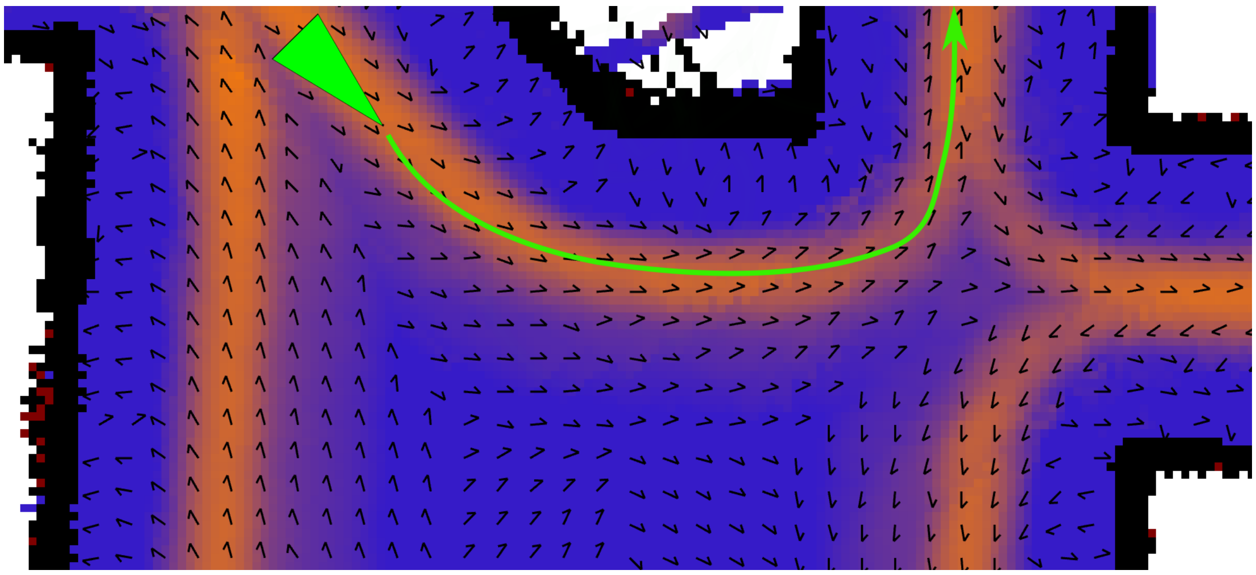


Figure 6.3: A map of showing the maximum reward achievable for a left wall follower at different positions in an environment, overlaid with vectors indicating the orientation the maximum was achieved at. Rewards fade from blue to orange, with blue indicating low reward states and orange indicating high. The green triangle and path denote a robot of realistic scale and the rough trajectory it would follow from the indicated starting pose. Note how the vectors direct the robot along the left wall, producing the desired behavior.

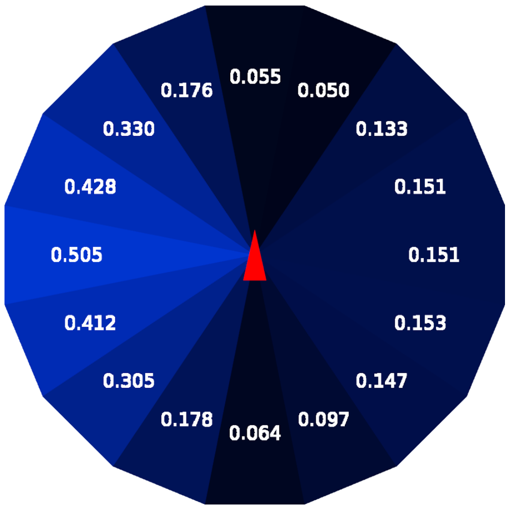
gradients for a variety of poses can be seen in Fig. 6.3, while an underlying heat map shows local variation in reward.

It is straightforward to numerically calculate the translational gradients. We compute this gradient by fixing the robot's orientation at optimal and inducing small translations in the viewpoint. The resulting gradient acts as a corrective force, guiding the robot towards local optimal in the reward function. For example, in the wall following tasks, this force acts keep the robot near a learned, ideal distance from the wall it is following. Because the scale of this gradient can vary widely in different portions of the environment, we find it useful to normalize it before use.

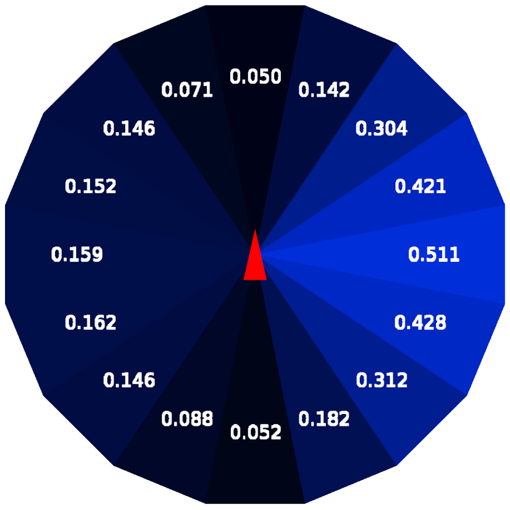
Though we described the process for computing the gradient at the current pose of the robot, in practice, it is useful to compute gradients at some small lookahead distance. This acts to dampen the resulting controller. It is straightforward to implement this by adding small corrections to the robot's range observations to account for the translation. In this work, we use a lookahead distance of 0.3 m from robot center, which corresponds with the front of our robot.

6.2.6 Weighting Features for Task Relevance

We have described a method for scoring a state based on the likelihood of observing the particular feature values making up that state. By extracting and following the gradient of this function, we



(a) Left Wall Following



(b) Right Wall Following

Figure 6.4: Feature weights learned through SGD, overlaid on the segments of the robot's sensor observations the features were constructed from. The learned weights show clear preference for features observing the wall the robot is following.

reproduce a demonstrated behavior. In addition to using features designed to capture important aspects of the state space for the task, it is common practice to learn feature weights based on the relative importance of features to each other. We employ this strategy to further improve policy performance.

Though our features encode important information about the similarity of a particular state to states observed during the demonstrations, there is still room for improvement. We can further optimize the policy reward function by adding feature weights, updating it to:

$$R(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{N} w_i \ln \Big(\sum_{j=1}^{K} \phi_j f(x_i | \mu_i^j, \sigma_i^j) \Big) \tag{6.6}$$

where $\boldsymbol{w}$ is the set of feature weights maximizing the reward function to produce the desired actions. These actions can be extracted from the demonstrations, which encode the actions taken by the demonstrator at each state.

Stochastic Gradient Descent (SGD) is a straightforward optimization technique which can be used to compute the weight vector $\boldsymbol{w}$. SGD is an iterative method which learns the parameters to some function based on a provided training set. The algorithm considers each example of the training set in turn, computing the gradient of the function and applying a small update to the parameters based on the gradient.

Typically, SGD employs a learning rate parameter η to control the size of the update steps. By reducing η over time, the algorithm will typically converge to either a global or local minimum of

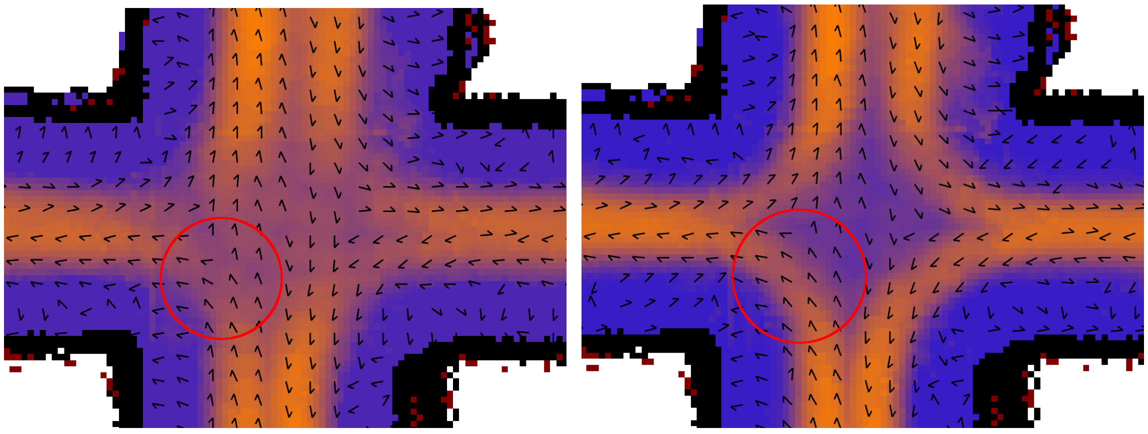

(a) Equal Feature Weights (b) Learned Feature Weights

Figure 6.5: Rewards and optimal orientation vectors for a right wall following behavior before and after learning feature weights. Blue indicates lower reward states, while orange indicates the highest reward states. The black vectors denote the optimal orientation for a given state. Note the areas circled in red. When the features are unweighted (a), turning left and going straight are relatively similar in reward, resulting in the robot sometimes taking the incorrect action. Once learned feature weights are incorporated (b), the features rewarding the incorrect action are penalized, resulting a tight channel of high reward guiding the robot correctly around the turn.

the function. It is common practice to randomize the order in which the training data is presented after each training epoch, which can mitigate the impact of patterns or local biases in the data. In our case, we present viewpoints and associated actions taken during the demonstrations in random order.

We use SGD to compute the optimal weight vector $\boldsymbol{w}$ based on the actions provided in the demonstrations. We present data in random order, sampling single training examples uniformly at random from the set of all demonstrations. We set our learning rate $\eta = 10^{-4}$. We impose an additional constraint $|\boldsymbol{w}| = 1$, which bounds the size of updates that can be made but otherwise does not impact the relative values of the learned weights. Visualization of our learned feature weights with respect to the portions of the robot's sensor observations they apply to can be seen in Fig. 6.4.

The learned weights play a key role in producing correct actions. An example situation in which the learned weights lead to a change in the produced behavior can be seen in Fig. 6.5, which visualizes the cost surface and orientation gradients for a right wall follower. When the features are equally weighted (Fig. 6.5a), a robot approaching from the right will fail the make the turn, only swerving slightly towards the turn before going straight. One the learned weights are added (Fig. 6.5b), the region in the middle of the intersection becomes more severely penalized, channeling the robot correctly around the turn.

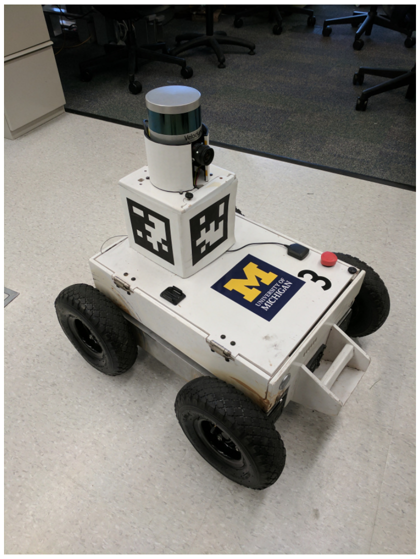


Figure 6.6: The MAGIC 2 robot, a custom platform built by the APRIL robotics lab at the University of Michigan. The primary sensor of the MAGIC 2 robot is the Velodyne Puck, a 3D LIDAR capable of collecting 300,000 range measurements per second. This robot was used to collect the data in this chapter.

6.3 Evaluation

We evaluate our method on two behaviors: left and right wall following. Wall following behaviors are well suited to navigation in many indoor environments, which largely consist of hallways leading to places of interest. By switching between left and right wall following modes at strategic moments, large-scale navigation plans can be constructed.

6.3.1 Data Collection and Training

Data was collected by logging teleoperated demonstrations of the desired behaviors in several buildings on the University of Michigan Engineering campus. The robot used in these experiments is the APRIL MAGIC 2 platform, pictured in Fig. 6.6. The primary sensor used for generating training examples is the robot's Velodyne Puck LIDAR, a ranging sensor with 16 IR lasers arranged in a vertical fan. Robot motion is estimated through a combination of wheel encoders and gyroscopic data from an inertial measurement unit.

In total, we collected 13,604 sample viewpoints of right wall following behavior and 13,391 of left wall following behavior across in two unique buildings, covering a total of 13 unique stretches of wall. Samples were collected at intervals of 100 ms, resulting in a total of 45 minutes of training data between both policies. All demonstrations were collected by the same operator.

All of the following results are presented for a feature space based on segmenting the robot's range measurements into 16 equally sized segments and storing the mean range observed by the LIDAR within that segment. Mixtures of 3 Gaussians are fit to describe the distributions of feature values. This parameter was manually tuned to exhibit the most consistent performance.

6.3.2 Test Environments

We evaluate our algorithm using synthetic range detections generated from maps of several real-world environments, pictured in Fig. 6.7. These 2D maps were generated from data collected by a MAGIC 2 robot and processed with an offline SLAM system. The maps have undergone small amounts of post-processing to add in glass walls removed by the automatic compositing algorithm.

Training data was collected in portions of the environments pictured in Figs. 6.7a and 6.7b. No training data was collected in the environment pictured in Fig. 6.7c, which was reserved strictly for evaluation purposes.

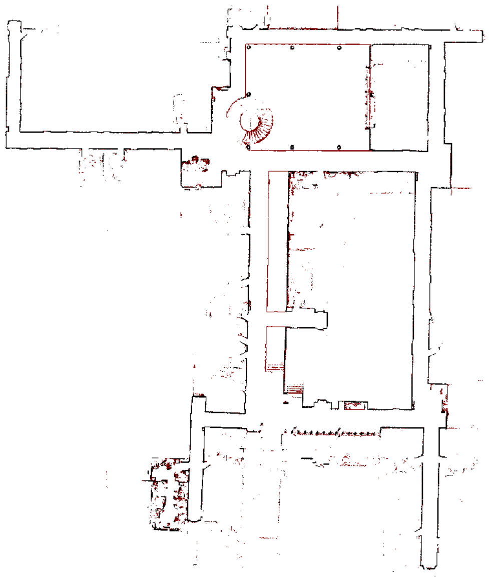
6.3.3 Behavior Reward Maps

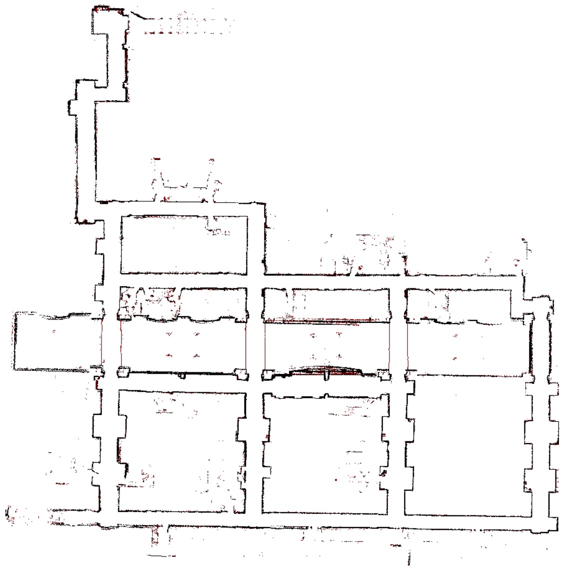
It is desirable for the learned reward functions to generalize well across environments, even if those environments were not used in training. By necessity, the learned rewards should also have distinct preferences, so that DART is provided with a sufficiently diverse selection of actions to cover the environment.

In our first evaluation, we investigate which states each reward function has learned to prefer. We present these results in the form of maximum reward maps on a scale from blue to orange, where blue indicates low-reward states and orange indicates the highest reward states.

To generate a map, we exhaustively generate synthetic viewpoints in the test environments. Viewpoints are generated only for areas of known free space, since these are locations a robot could theoretically travel to. Viewpoints are generated every 5 cm within this space (at the same resolution as maps). Since orientation has a large impact on the computed reward, we evaluate the reward function multiple times for each location, constructing feature vectors for viewpoints rotated by 5 degree increments. In lieu of displaying a separate map for each possible orientation, we present a single, unified map displaying the maximum reward seen for any orientation. Call outs at interesting locations present zoomed-in views overlaid with vector fields indicating the orientations at which maximum rewards were obtained. These vector fields are useful in illustrating the differences in preferences learned by each policy.

As can be seen in Figs. 6.8, 6.9, and 6.10, the learned policies demonstrate clear preferences for poses near walls on the appropriate side of the robot. The result is the formation of luge-like troughs centered on the highest scoring traits, suggesting the learned function will be suitable for integration in our proposed control framework.


(a) 3rd Floor BBB


(b) 2nd Floor EECS

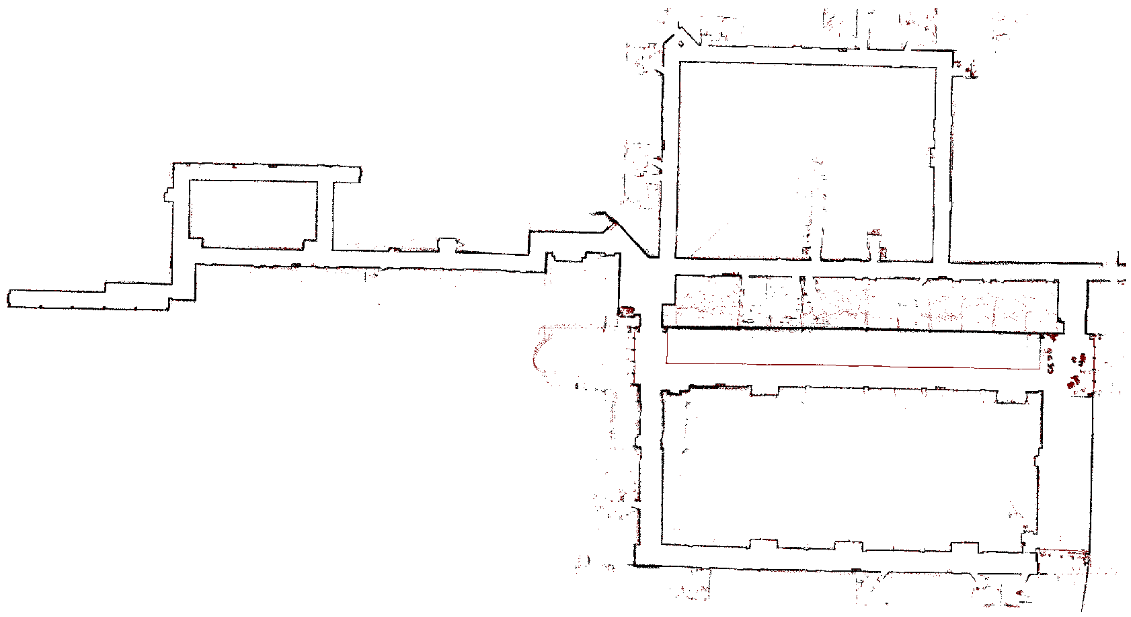

(c) 3rd Floor GG Brown

Figure 6.7: Maps of test environments used in evaluation. Black and red indicate structure, which black structure indicating highly vertical structure and red indicating all other structure that would hinder navigation.

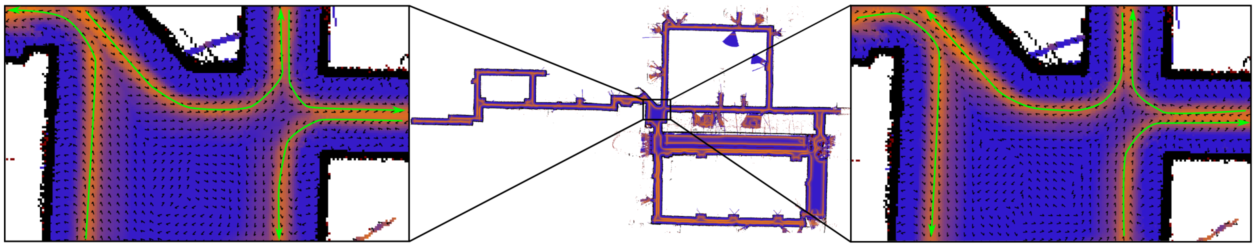


Figure 6.8: A visual representation of high reward states for left and right wall following policies in the GG Brown Building. Higher reward states are shown in orange, while lower reward states are shown in blue. Black vectors in the call-outs indicate the optimal orientation of the robot at the noted location for left vs. right wall following, while green arrows indicate the typical trajectories followed by the robot for the given policy.

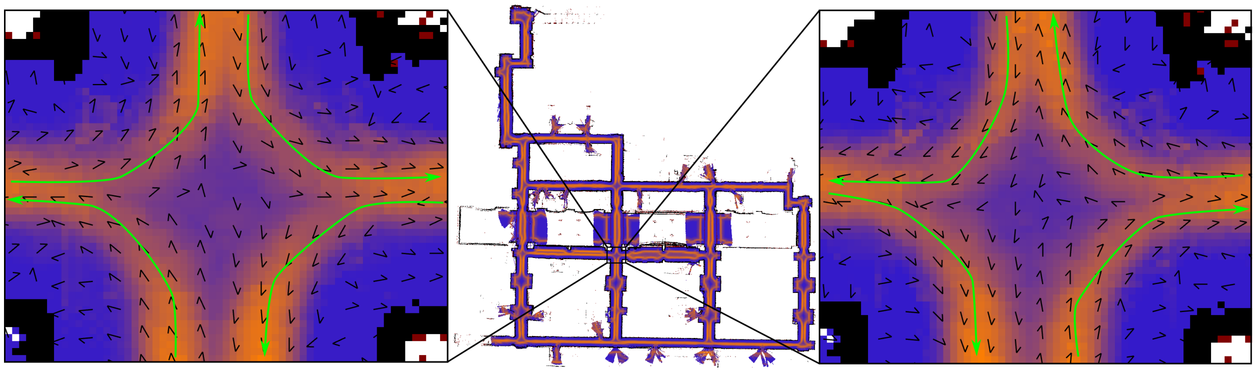


Figure 6.9: A visual representation of high reward states for left and right wall following policies in the EECS building. Higher reward states are shown in orange, while lower reward states are shown in blue. Black vectors in the call-outs indicate the optimal orientation of the robot at the noted location for left vs. right wall following, while green arrows indicate the typical trajectories followed by the robot for the given policy.

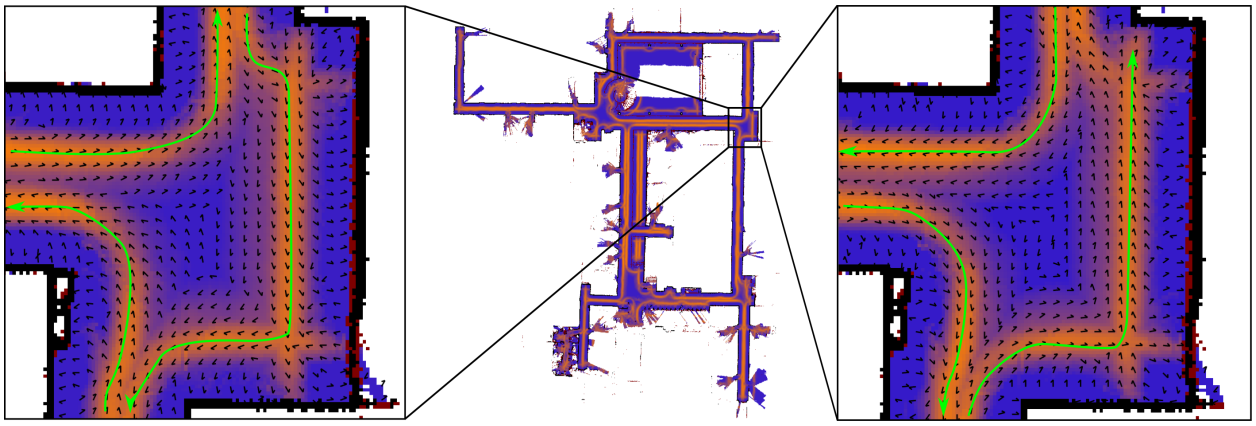


Figure 6.10: A visual representation of high reward states for left and right wall following policies in the BBB building. Higher reward states are shown in orange, while lower reward states are shown in blue. Black vectors in the call-outs indicate the optimal orientation of the robot at the noted location for left vs. right wall following, while green arrows indicate the typical trajectories followed by the robot for the given policy.

6.3.4 Behaviors in Action

To support DART, the learned behaviors should possess two main characteristics. Foremost, the behaviors must be capable of generating different routes from the same start state; this ensures that DART can generate a diverse set of routes to all corners of the environment. Behaviors should also be repeatable; while DART can account for types and varying frequencies of failures, it is best suited to domains where behaviors *generally* perform reliably. DART's strength comes from detecting the portions of the environment where particular sensing or navigation strategies are weak and adapting to use more reliable strategies or avoiding the region whenever possible.

In this section, we demonstrate these characteristics based on trajectories collected in simulation. We perform these simulations based on maps built from traversals of real environments. Synthetic LIDAR returns perturbed by 1% noise are generated based on the robot's current position in the environment. This level of noise is consistent with the performance of a real LIDAR sensor. As in the previous section, we perform tests in three environments, one of which was not used to when training behaviors.

To test reproducibility, we run 20 simulations of each control policy from similar starting poses, perturbing the pose slightly. The goal of these perturbations is to demonstrate the robustness of the policies to slight variations in starting state. We expect the learned reward functions to form wide basins of convergence in the environment, drawing the robot quickly into steady, repeatable trajectories. We also expect the learned policies to take different actions at key decision points, demonstrating that the policies can produce functionally different routes. We present numerous examples from key decision points in the environment, showing that the behaviors meet these expectations, taking correct and consistent actions in the general case.

Figure 6.11 presents several representative examples of combinations of consistency and semantic correctness exhibited during our evaluation. We indicate the trajectories taken by simulated robots from a variety of starting configurations and annotate the decision point with arrows indicating the semantically correct actions to take. The color and patterns of these arrows indicate the characteristics exhibited by the robot when executing the learned policy. A solid green arrow indicates the best possible outcome: a behavior was followed consistently and produced semantically correct actions. A red dotted arrow indicates a behavior which, while consistent, produced semantically incorrect action such as passing by a hallway which the robot should have turned down. Finally, a dark green dashed arrow indicates a behavior that exhibited a mixture of correct and incorrect actions.

While the results shown in Fig. 6.11 show only the outcomes for a single direction of approach to a key decision point, in subsequent figures we present composite results indicating the outcomes of trials for all possible directions of approach to the area of interest. In cases where incorrect actions are taken, this can result in multiple directions of approach converging on the same stable

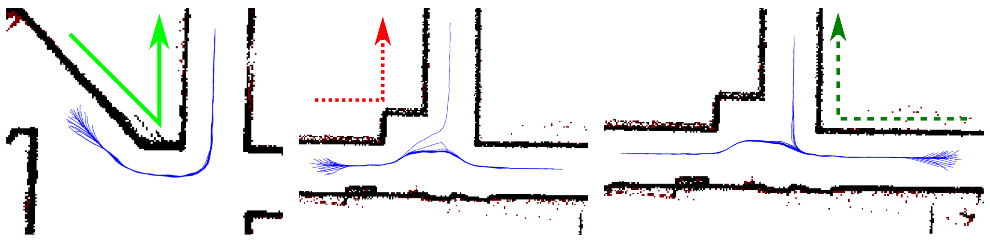

(a) Correct, Consistent (b) Incorrect, Consistent (c) Inconsistent

Figure 6.11: Example trajectories and instances of different characteristics exhibited by learned behaviors. Blue lines show trajectories produced by simulated trials, while colored arrows indicate the semantically correct behavior the trajectories should produce. In the majority of cases, behaviors should produce semantically correct and (mostly) consistent actions (a). This is critical to enabling the robot to take a wide variety of actions at decision points. However if in rare instances actions are consistently incorrect (b), DART can still use the reliability of these actions to produce robust plans. Inconsistent actions (c) are least desirable, since the robot cannot be relied upon to follow the intended trajectory.

final trajectory.

As expected, the learned policies show great levels of consistency, converging from a variety of starting states to near-identical trajectories. Once converged, behaviors typically executed trajectories repeatable within 10 cm. Example trajectories of left wall following in areas of interest within the test environments can be seen in Figs. 6.13, 6.15, and 6.17. Similar results for a right wall following policy can be seen in Figs. 6.14, 6.16, and 6.18.

We classify an execution as "semantically correct" if the robot follows a qualitatively similar trajectory to that a human operator would take. To evaluate these actions, we evaluate the robot as it enters areas with key decision points in the environment, which we designate as intersections between corridors. In 76% of the scenarios tested, the simulated agent consistently took the expected action for the behavior in question, exhibiting both semantically correct and repeatable behavior. The simulated agent consistently took a semantically incorrect action another 16% of the time. In the remaining 8% of scenarios, the robot took some mixture of semantically correct and incorrect actions.

The most common causes of semantically incorrect actions are local minima, which result in the robot becoming stuck, and narrow corridors, which the robot sometimes skips over rather than turning into. Local minima are particularly problematic at the ends of narrow corridors. Though the semantically correct action would be to turn around at the end of the corridor and continue following the wall, the greedy nature of the potential field algorithm consistently prevents the robot from exploring this action.

While this means there is room for improvement in the semantic accuracy of the policies, the current level of accuracy and high levels of repeatability suggest that the learned behaviors are well

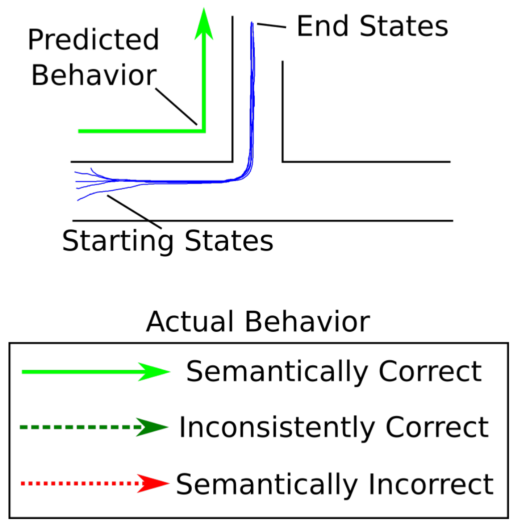


Figure 6.12: Legend for Fig. 6.13 - Fig. 6.18.

suited for use in a DART framework. In the rare instance of inconsistent executions, DART can observe the failure mode and adapt its plans accordingly to avoid the unreliable behavior.

6.4 Summary

DART requires a set of easily-simulated, closed-loop policies to construct policy chains describing routes through the environment. In this chapter, we propose that controllers for suitable behaviors can be learned from demonstrations of desired policy. Demonstrations are an appealing source of training data, as they can be provided by experts and non-experts alike.

Our method involves learning a reward function to integrate in a potential-field style framework. This allows us to combine learned preferences for behavior actions with repulsive actions, ensuring that the resulting policy will be both correct for the task and collision-free. Our method fits distributions to statistic features describing the robot sensor states observed during demonstrations and uses them to evaluate the likelihood of observing novel states. We reproduce the demonstrated behavior by greedily taking actions which draw the robot to high scoring states, where higher scores correspond with greater similarity to states observed during the demonstrations.

The learned policies produce semantically similar behaviors to the demonstrated policies, indicating the validity of our approach. Additionally, the resulting policies are highly repeatable, making them well suited for use in the DART framework. Our method is able to compute new policies quickly from easy-to-collect demonstration data, making it easy to learn new policies on the fly or extend existing policies with additional data, offering opportunities for non-expert users to modify systems for their specific environments or tasks.

Behavior: Left Wall
Environment: EECS

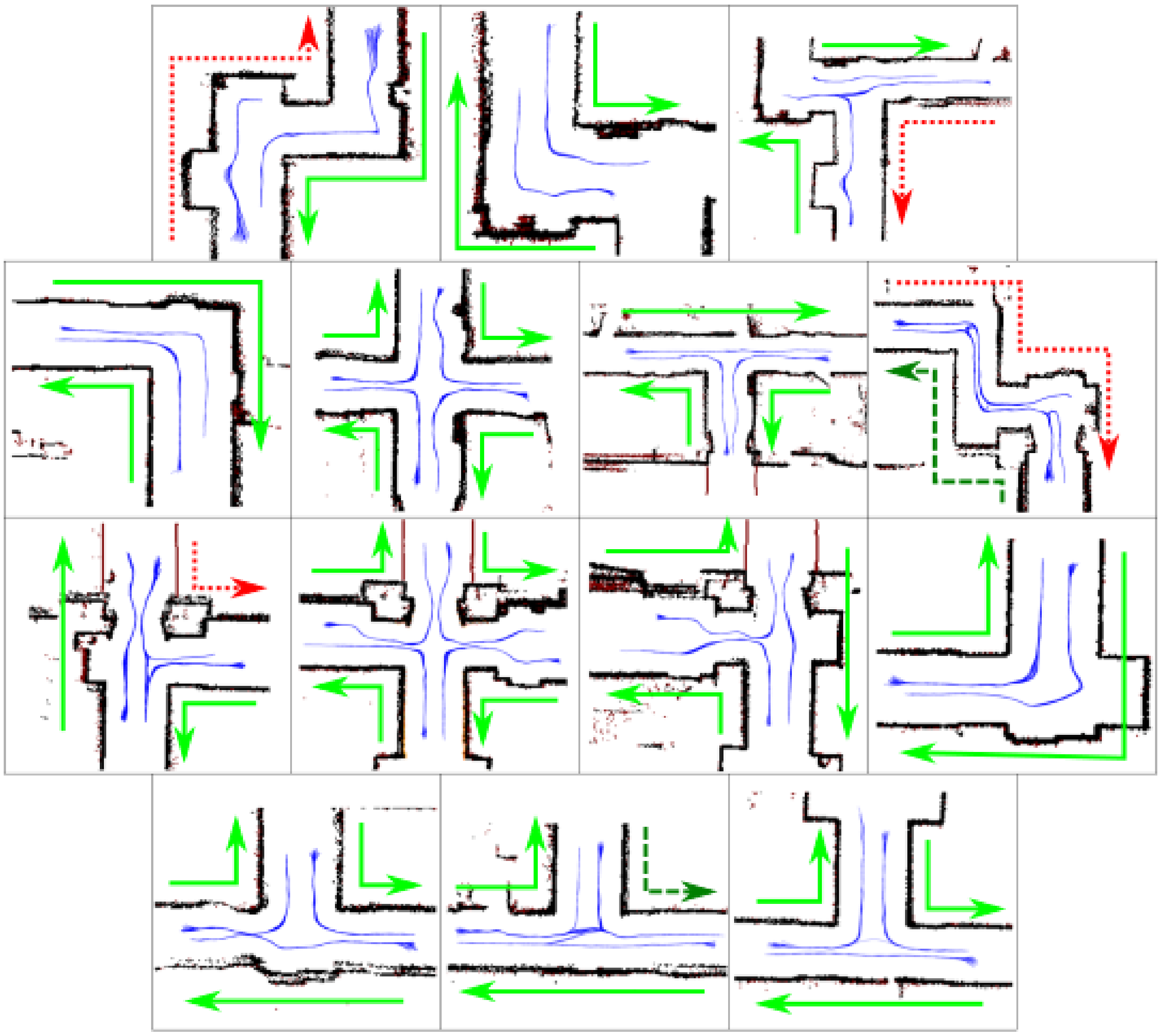


Figure 6.13: Left wall following trajectories in EECS. Blue lines show trajectories produced by simulated trials, arrows indicate the predicted behaviors, and arrow colors indicate quality of the actual behavior. The learned policy produces consistent, repeatable behaviors which generally take semantically correct actions.

Behavior: Right Wall
Environment: EECS

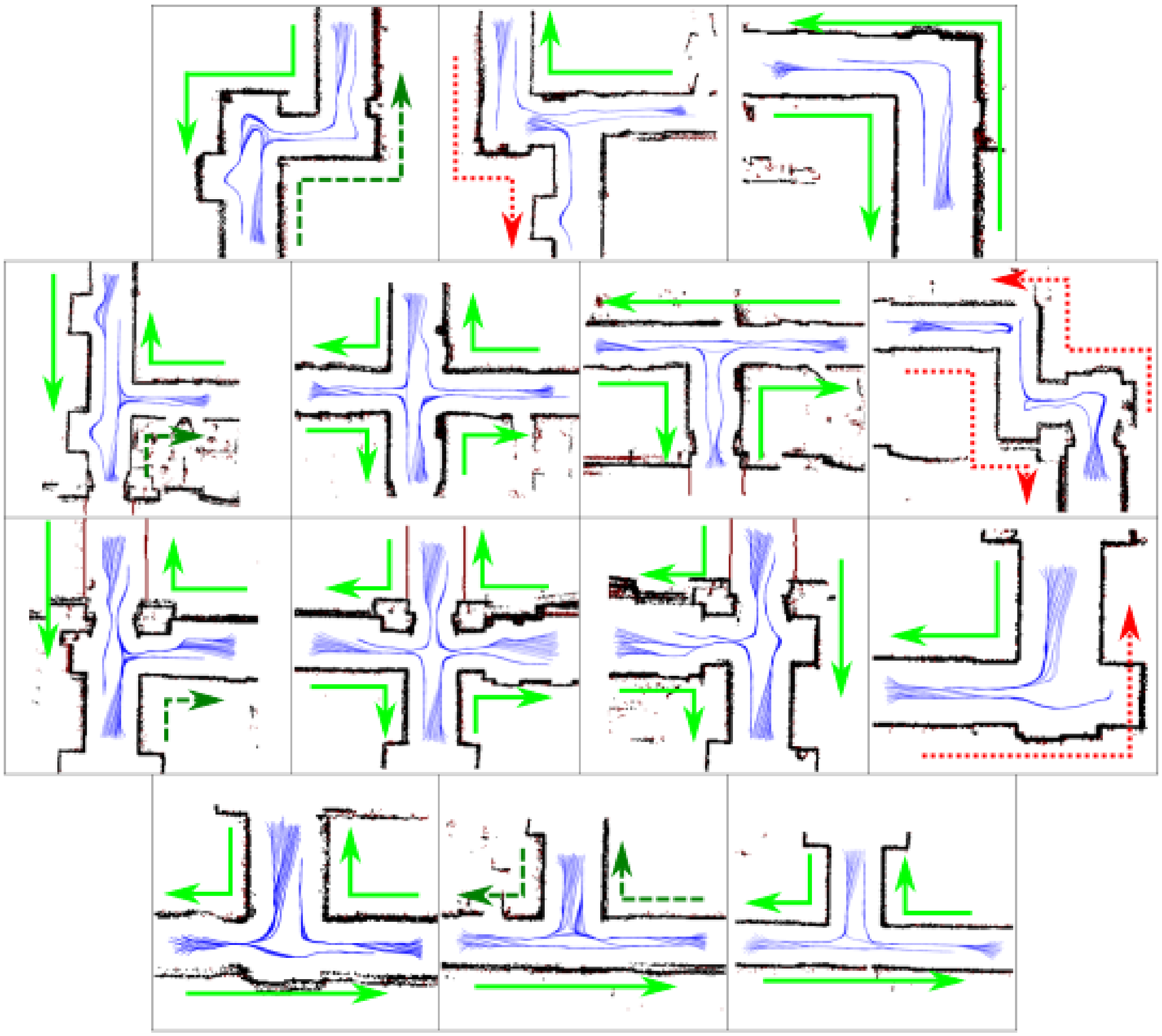


Figure 6.14: Right wall following trajectories in EECS. Blue lines show trajectories produced by simulated trials, arrows indicate the predicted behaviors, and arrow colors indicate quality of the actual behavior. The learned policy produces consistent, repeatable behaviors which generally take semantically correct actions.

Behavior: Left Wall
Environment: BBB

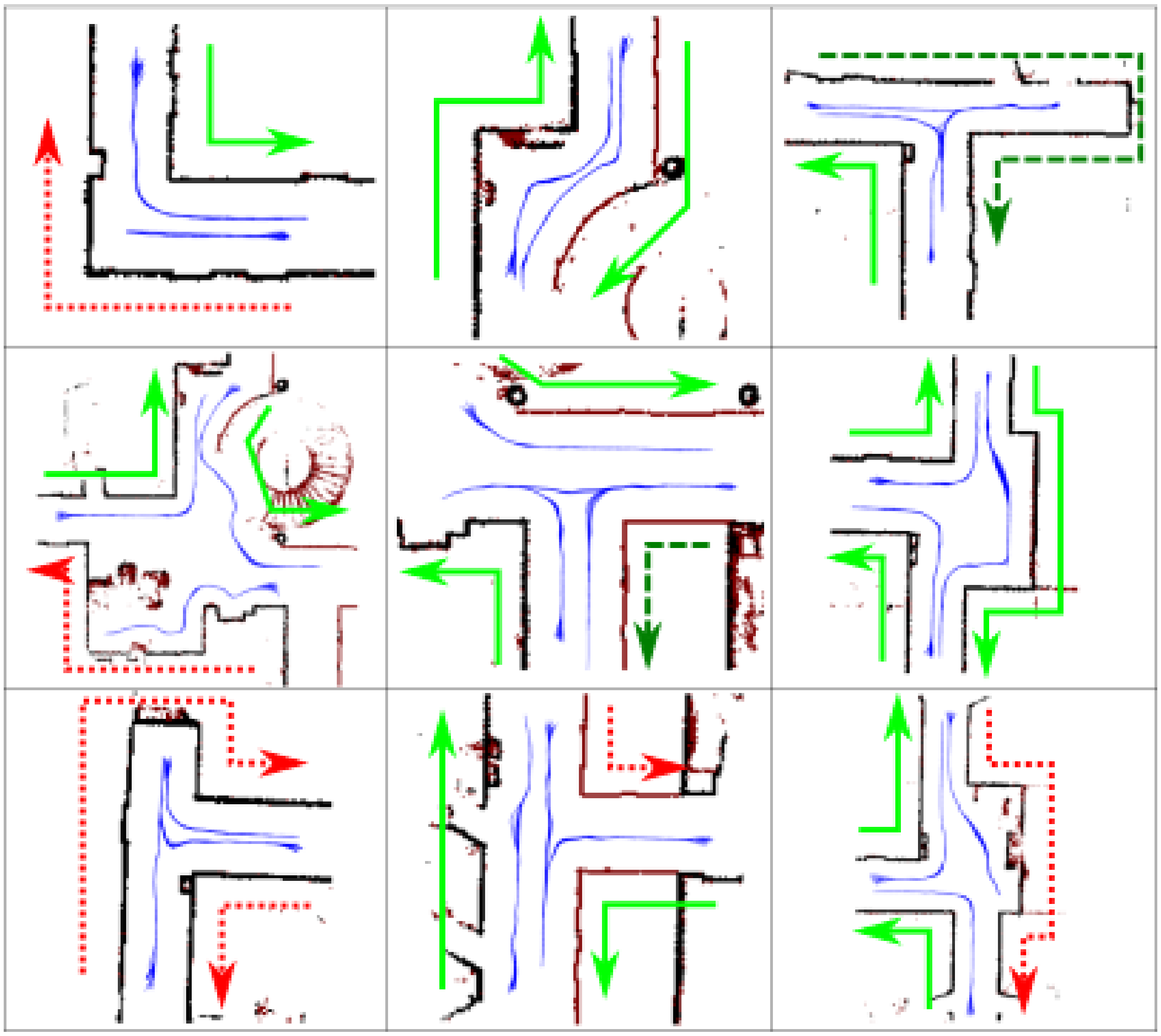


Figure 6.15: Left wall following trajectories in BBB. Blue lines show trajectories produced by simulated trials, arrows indicate the predicted behaviors, and arrow colors indicate quality of the actual behavior. The learned policy produces consistent, repeatable behaviors which generally take semantically correct actions.

Behavior: Right Wall
Environment: BBB

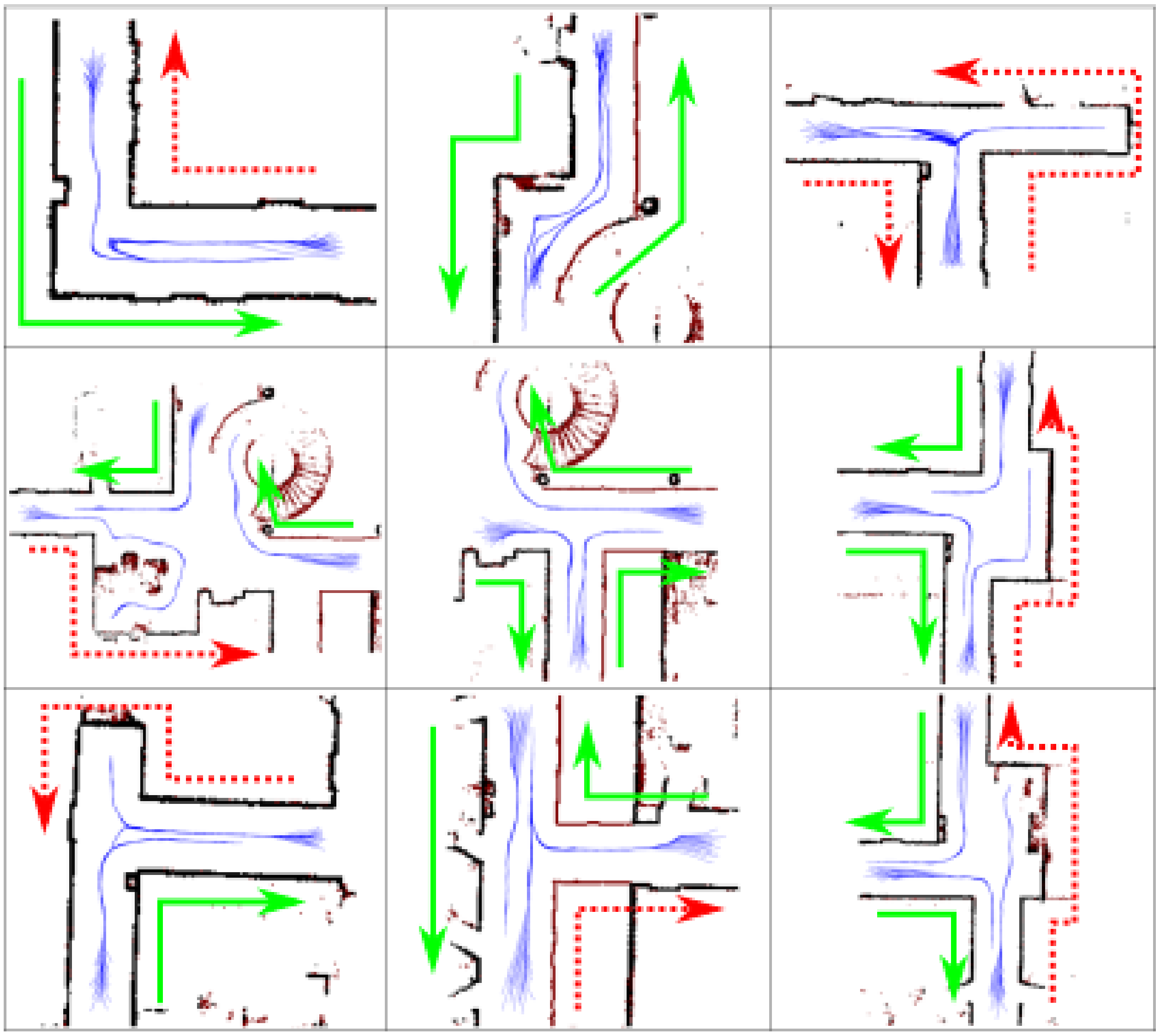


Figure 6.16: Right wall following trajectories in BBB. Blue lines show trajectories produced by simulated trials, arrows indicate the predicted behaviors, and arrow colors indicate quality of the actual behavior. The learned policy produces consistent, repeatable behaviors which generally take semantically correct actions.

Behavior: Left Wall
Environment: GG Brown

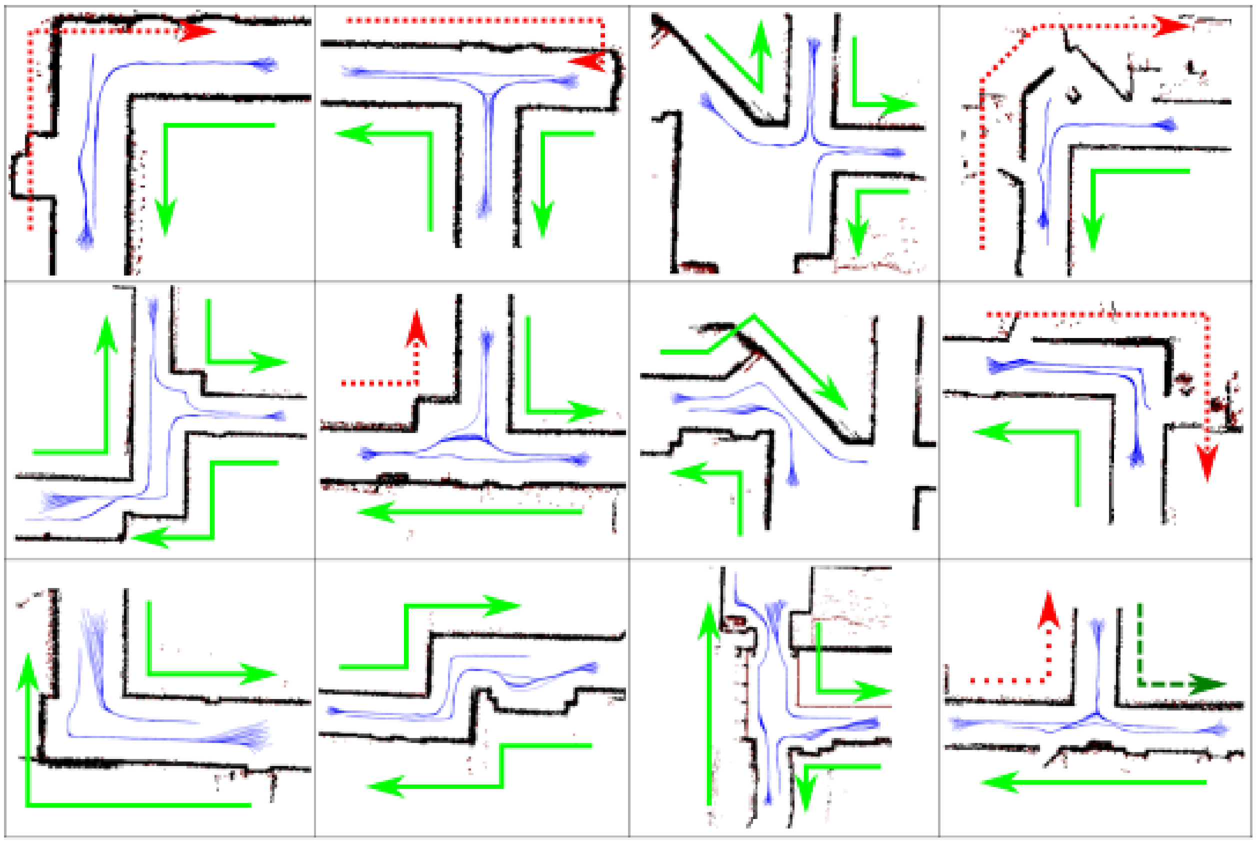


Figure 6.17: Left wall following trajectories in GG Brown. Blue lines show trajectories produced by simulated trials, arrows indicate the predicted behaviors, and arrow colors indicate quality of the actual behavior. The learned policy produces consistent, repeatable behaviors which generally take semantically correct actions.

Behavior: Right Wall
Environment: GG Brown

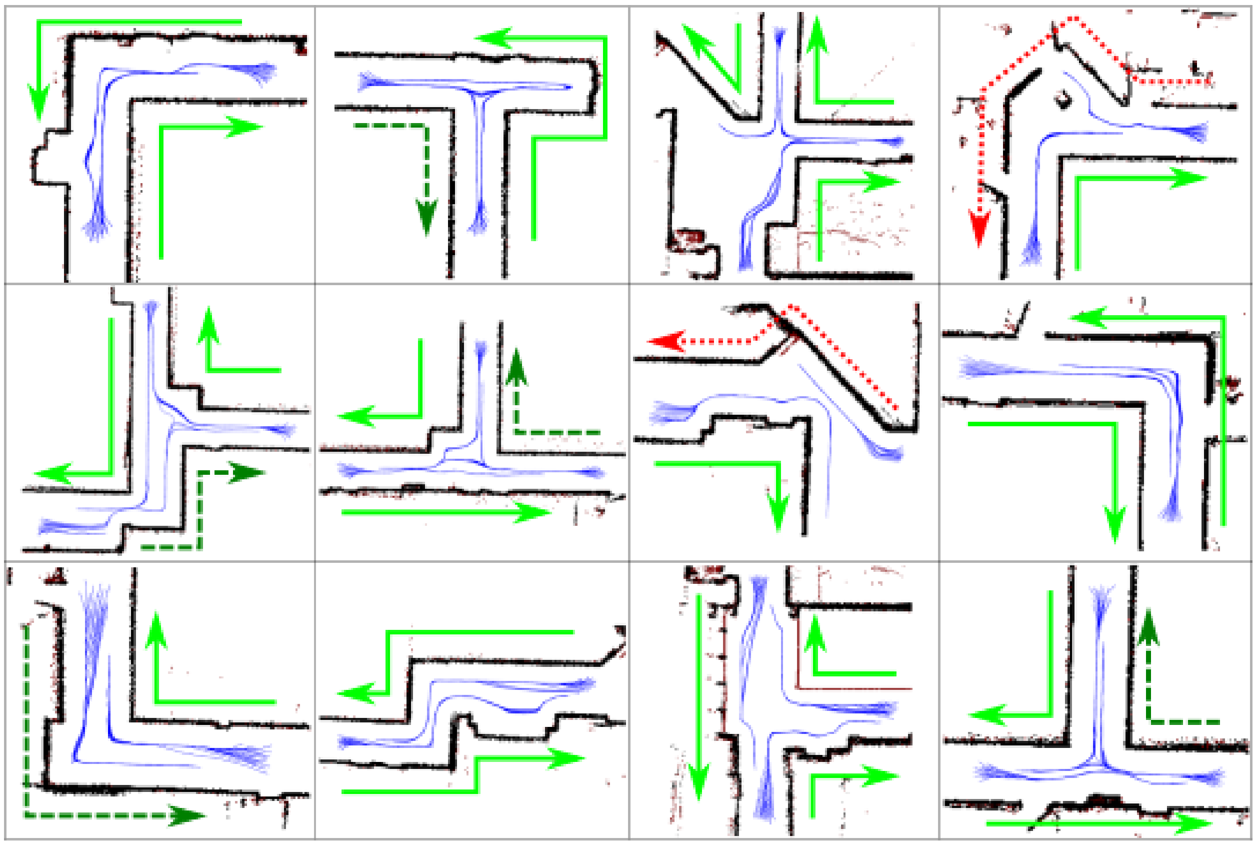


Figure 6.18: Right wall following trajectories in GG Brown. Blue lines show trajectories produced by simulated trials, arrows indicate the predicted behaviors, and arrow colors indicate quality of the actual behavior. The learned policy produces consistent, repeatable behaviors which generally take semantically correct actions.

CHAPTER 7

Conclusion

Navigation is a fundamental robotic capability on top of which many high-level tasks are built. It is important that these underlying navigation systems provide a solid foundation, demonstrating robustness sensing and actuation and actuation errors while remaining easily extensible to new environments. In this manner, strong navigation capabilities allow robots to perform a wide variety of tasks safely and reliably.

Robotic systems are typically equipped with powerful, general-purpose navigation systems suitable for a wide variety of tasks. This generality comes with drawbacks, however, enforcing strong needs for localization and accuracy in control that are not always necessary to accomplish the desired task. The key observation in this thesis is that sometimes, simple, less general navigation strategies are sufficient for specific tasks and are more robust to possible errors. The question we address is: when is it appropriate to use these methods in lieu of more general ones to take advantage of this robustness?

Too commonly, navigation systems ignore the problem of reliability entirely, instead relying on human experts to "rescue" the robot in the unlikely event of failure. For small scale applications, this strategy may be sufficient; in an adequately controlled environment, the chance of failure may be minimal and human assistance can always be nearby. However, as operational domains become larger and more robots make their way into the world, interventions become more costly, requiring larger numbers of expert technicians covering larger regions of the environment. Though some failures (such as mechanical ones) are inevitable, by incorporating reliability into the navigation planning paradigm, we can drive down the rate of errors due to imperfect execution, minimizing the number of rescue interventions necessary.

7.1 Contributions

The primary contribution of this thesis is a planning framework which considers the reliability of the actions available to the robot when selecting routes through the environment. In the first half of this thesis we introduce DART, a navigation framework employing Monte Carlo simulations to evaluate the distribution of outcomes of actions available to the robot. From the results of these simulations, we are able to construct and select policy chains describing a sequence of actions to take which optimize both for the efficiency of the route as well as its repeatability. We initially introduce this framework in the context of providing directions to a modeled wayfinder, and then show more concretely how the concepts can be applied to robotic domains.

Policy chains are a powerful tool for compactly describing routes for the robot to follow. They allow us to describe routes in terms of macro-actions (e.g. ”Follow the wall until the T-intersection”), avoiding unnecessary levels of specificity (e.g. describing a 100 m path in 10 cm steps). Policy chains are composed of two components: closed-loop control policies describing ways in which the robot can move about the environment and switching conditions which can be used to trigger changes in policy. Planning in the space of policy chains allows DART to selectively change navigation strategies based on their effectiveness in different portions of the environment.

DART's viability is dependent on the existence of these underlying building blocks. In the second half of this thesis, we tackle the problem of constructing suitable control policies and termination conditions for employing DART-style navigation on a robot. A key challenge is that these components must be easily simulatable, so that DART may produce realistic estimates of the outcomes of trying different actions. We focus our efforts on techniques based upon range sensors, as they are commonly found on most modern robots and their measurements are easily simulated in maps produced by SLAM systems.

We contribute two methods appropriate for use in DART. First, we present a method for identifying landmarks in the environment. We describe a network structure and training techniques to teach a CNN to classify places semantically from 2D range data. Our method is suitable for describing classes of places such as "room", "corridor", or "intersection." These types of places are of interest, as they often correspond with key decision points along routes.

Second, we contribute a method for teaching a robot control policies from navigation. Our method uses teleoperated demonstration data to build a reward function scoring states based on their similarity to those encountered during demonstrations. We use this reward function to define a behavior function suitable for integration within a potential field framework, allowing us to reproduce the demonstrated behaviors in simulation. The resulting behaviors produce consistent and correct actions for wide varieties of starting states in several environments, indicating that the learned policies generalize well.

Both our learned policies and place classifier consume 2D range data as their primary input; a form of data commonly available to modern robots. This data is easily synthesized, a critical requirement for use in DART, through raycasting operations on SLAM maps.

7.2 Future Work

The methods described in this thesis allow robots to incorporate reliability into their navigational planning considerations, resulting in plans that avoid challenging portions of the environment. Our methods rely on availability of robust control policies and termination conditions to build upon, and we suggest two possible ways to supply these capabilities to the robot. While this thesis provides the foundation for building robust navigation systems, there are several opportunities for improvement.

7.2.1 Detecting and Recovering From Bad Actions

Chapter 3 poses the problem of robot navigation as the problem of giving high-quality directions. For our purposes, directions are defined as a sequence of actions to take, in order, to arrive at one's destination. While DART will discover the most robust sequence of actions available to the robot, this does not guarantee that the robot will execute the final plan without error. These failures are no less catastrophic to DART than any other planning method, motivating the addition of recovery strategies to handle these failures.

The first step in recovering from error is to detect that an error has occurred. The robot can detect errors in execution by comparing its observations during plan execution to the expected observations from DART's simulations. Sometimes this ma be easy, as when the robot passes a landmark that could only have been seen during an incorrect execution. In more challenging cases, the robot may have to consider the shape of its trajectory or timing of its actions compared to an ideal execution.

Currently, when an error is detected our strategy is to plan a new policy chain from scratch. However, in practice, directions often incorporate extra cues for use in recovery operations. For example, if an intersection is particularly easy to miss, the direction-giver might add a note to the end of the instruction. For example, "Go until you reach an intersection with a big tree on the corner. If you reach the bridge, you've gone too far."

Incorporating recovery cues into planning is a powerful way to increase the reliability a set of directions. Though largely unexplored in this work, these cues might be incorporated into the DART framework by allowing policy chains to branch and reconverge. One strategy for representing this type of action is to extend our search to the space of policy-trees, where branches of the trees encode recovery actions for likely mistakes. However, this extension shares many of the same challenges as those discussed in Sec. 3.3.2, in particular posing large computational issues.

7.2.2 Determining the Optimal Set of Building Blocks

The selection of control policies and termination conditions play a deep role in the types of routes produced by the robot. If the suite of capabilities available to the robot is insufficiently diverse, portions of the environment may remain unreachable to the robot. This thesis proposes that, in the particular instance of indoor environments, left and right wall following should provide good coverage, but does not explore the question in great detail for more complicated settings. As robots move into more diverse environments, it will become necessary to rigorously define what constitutes a sufficient *covering* of the environment.

While ensuring coverage of the environment is important, is is also of interest to select the minimal set of control policies and switching conditions that will allow the robot to fully and effectively navigate the environment. DART suffers a similar weakness to all planning algorithms: as the branching factor becomes larger, it becomes more expensive to search the space of possible actions. However, a more diverse set of actions allows DART to more effectively adapt to the challenges posed by the environment. Future work should evaluate the trade offs between computational efficiency and plan quality based on the selected actions.

7.2.3 Accuracy of Simulation

The success of DART is predicated on the accuracy of the underlying simulations of available robot actions. One of the key questions for DART-style navigation is: what level of simulation is necessary to realistically estimate the outcome of an action? Determining the appropriate balance between computation and the accuracy of simulations will be critical to adapting our method to new domains.

If the simulations produce sufficiently different motions or outcomes from those produced by the target platform for the desired policies, DART loses the ability to reliably estimate the outcomes of policy-chains. Characterizing the impact on the efficacy of DART-style planning in the presence of varying levels of simulation accuracy is important to determining which types of policies and conditions are reasonable to support.

For example, the wall following behavior presented in Ch. 6 evaluation produced stable behavior for several different parameterizations of the underlying platform (e.g. varying maximum drive speeds, amounts of internal dampening, and inertia). However, once the parameters deviated sufficiently from those of the actual robot platform used in the demonstrations, the learned behaviors show some signs of instability. While ideally, simulation would perfectly model the capabilities of the real robot, in reality, this is often an unrealistic expectation. Instead, it is interesting to consider which types of policies would be most robust to inaccuracies or variation in the robot model. These policies seem most likely to be effective building blocks for DART.

Figure 7.1: The author telling a robot where it is supposed to go. Human friendly interfaces based on language could improve the experience of interacting with robots, allowing wider audiences access to this technology.

7.2.4 Increasing Simulation Complexity

As environments for operation become larger and more complex, it may also become challenging to model and run the necessary simulations to determine the effectiveness of hypothetical actions. The scope of this thesis is limited to single-building, indoor applications, but automated vehicles, for example, may operate in regions covering many square miles, where weather, traffic, and road conditions can play large roles in which navigation strategies prove most effective.

Computational challenges are exacerbated as interactions between multiple agents become important. It may be computationally infeasible to accurately approximate the space of possible interactions and their effect on policy selections. One possibility is to bias the sampling process towards the selection of more informative or influential samples, allowing the system to quickly identify and characterize high cost outcomes.

This strategy has been employed by Mehta, Ferrer, and Olson in their work on Multi-Policy Decision Making (MPDM) [67], a framework for control policy selection based on the predicted interactions between agents. In this work, the authors are able to efficiently discover critical interactions between agents that would result in undesirable outcomes (e.g. collisions) and adjust policies in real-time to avoid those scenarios. It is possible that similar biased-sampling techniques could be applied to the route-finding problem posed in this thesis, reducing computational requirements.

7.2.5 Human Interaction

When interaction with and alongside humans, it is desirable that robots act in predictable and understandable ways. A robot pursuing the shortest path may be efficiently accomplishing it's goal, but it is often difficult for the robot the express that goal, and it may come across as pushy if it cuts off some person to reach its goal more quickly.

Control policies and switching conditions can frequently be described with language, offering opportunities for human-robot-interfaces. In this thesis, we intentionally propose methods for learning policies and conditions with clear semantic meaning, leaving open the possibility for future efforts to make intuitive linguistic interfaces.

The primary advantage of semantically labeled actions lies in communication: robots can describe routes in human terms, allowing them to direct people around the environment. It also enables the robot to receive directions from a person, bypassing the planning process entirely. Linguistic interfaces offer an opportunity to make interactions with robots seem more natural and familiar.

A robot's core abilities also define how it will be perceived by the humans around it. While a shortest path follower might appear pushy, robots which stay to the sides of halls and avoid getting too close to people might be easier to anthropomorphize and accept. An interesting area of future research would be to evaluate policies not just based on their reliability, but on their perceived levels of politeness and intelligence. This information could allow systems to prefer actions which are less likely to offend nearby people.

BIBLIOGRAPHY

[1] ABBEEL, P., COATES, A., AND NG, A. Y. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research 29*, 13 (2010), 1608–1639.

[2] AKGUN, B., CAKMAK, M., JIANG, K., AND THOMAZ, A. L. Keyframe-based learning from demonstration. *International Journal of Social Robotics 4*, 4 (2012), 343–355.

[3] ALTHOFF, D., KUFFNER, J. J., WOLLHERR, D., AND BUSS, M. Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Autonomous Robots 32*, 3 (2012), 285–302.

[4] ANTONELLI, G., CHIAVERINI, S., AND FUSCO, G. A fuzzy-logic-based approach for mobile robot path tracking. *Fuzzy Systems, IEEE Transactions on 15*, 2 (2007), 211–221.

[5] ARGALL, B. D., CHERNOVA, S., VELOSO, M., AND BROWNING, B. A survey of robot learning from demonstration. *Robotics and autonomous systems 57*, 5 (2009), 469–483.

[6] BAGNELL, J. A., AND SCHNEIDER, J. G. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on* (2001), vol. 2, IEEE, pp. 1615–1620.

[7] BARRAQUAND, J., LANGLOIS, B., AND LATOMBE, J.-C. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics 22*, 2 (1992), 224–241.

[8] BEESON, P., JONG, N. K., AND KUIPERS, B. Towards autonomous topological place detection using the extended voronoi graph. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on* (2005), IEEE, pp. 4373–4379.

[9] BESL, P. J., MCKAY, N. D., ET AL. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence 14*, 2 (1992), 239–256.

[10] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[11] BROWNING, B., XU, L., AND VELOSO, M. Skill acquisition and use for a dynamically-balancing soccer robot. In *AAAI* (2004), pp. 599–604.

[12] BURNETT, G., SMITH, D., AND MAY, A. Supporting the navigation task: characteristics of 'good' landmarks. In *Proceedings of the Annual Conference of the Ergonomics Society* (November 2001), pp. 441–446.

[13] CALINON, S., AND BILLARD, A. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction* (2007), ACM, pp. 255–262.

[14] CANDIDO, S., AND HUTCHINSON, S. Minimum uncertainty robot navigation using information-guided pomdp planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (2011), IEEE, pp. 6102–6108.

[15] COHEN, B. J., CHITTA, S., AND LIKHACHEV, M. Search-based planning for manipulation with motion primitives. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on* (2010), IEEE, pp. 2902–2908.

[16] COHEN, R., BALDWIN, L. M., AND SHERMAN, R. C. Cognitive Maps of a Naturalistic Setting. *Child Development 49*, 4 (1978), pp. 1216–1218.

[17] DENIS, M., PAZZAGLIA, F., CORNOLDI, C., AND BERTOLO, L. Spatial discourse and navigation: an analysis of route directions in the city of Venice. *Applied Cognitive Psychology 13*, 2 (1999), 145–174.

[18] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[19] DONALD, B., XAVIER, P., CANNY, J., AND REIF, J. Kinodynamic motion planning. *Journal of the ACM (JACM) 40*, 5 (1993), 1048–1066.

[20] DUCKHAM, M., AND KULIK, L. "Simplest" Paths: Automated Route Selection for Navigation. In *COSIT* (2003), pp. 169–185.

[21] ELLIOTT, R. J., AND LESK, M. Route Finding in Street Maps by Computers and People. In *AAAI* (1982), pp. 258–261.

[22] EVTIMOV, I., EYKHOLT, K., FERNANDES, E., KOHNO, T., LI, B., PRAKASH, A., RAHMATI, A., AND SONG, D. Robust Physical-World Attacks on Deep Learning Models. In *arXiv preprint 1707.08945* (2017).

[23] FERGUSON, D., KALRA, N., AND STENTZ, A. Replanning with RRTs. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on* (2006), IEEE, pp. 1243–1248.

[24] FERGUSON, D., AND STENTZ, A. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics 23*, 2 (2006), 79–101.

[25] FIORINI, P., AND SHILLER, Z. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research 17*, 7 (1998), 760–772.

[26] Fox, D., Thrun, S., Burgard, W., and Dellaert, F. Particle filters for mobile robot localization. In *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 401–428.

[27] Friedman, S., Pasula, H., and Fox, D. Voronoi random fields: Extracting topological structure of indoor environments via place labeling. In *IJCAI* (2007), vol. 7, pp. 2109–2114.

[28] Ge, S. S., and Cui, Y. J. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation 16*, 5 (2000), 615–620.

[29] Ge, S. S., and Cui, Y. J. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots 13*, 3 (2002), 207–222.

[30] Goeddel, R., Kershaw, C., Serafin, J., and Olson, E. FLAT2D: Fast localization from approximate transformation into 2D. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (October 2016).

[31] Goeddel, R., and Olson, E. Dart: A particle-based method for generating easy-to-follow directions. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (2012), IEEE, pp. 1213–1219.

[32] Grollman, D. H., and Jenkins, O. C. Incremental learning of subtasks from unsegmented demonstration. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on* (2010), IEEE, pp. 261–266.

[33] Hahnel, D., Burgard, W., Fox, D., and Thrun, S. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* (2003), vol. 1, IEEE, pp. 206–211.

[34] Haque, S., Kulik, L., and Klippel, A. Algorithms for Reliable Navigation and Wayfinding. In *Spatial Cognition V Reasoning, Action, Interaction*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, pp. 308–326.

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

[36] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)* (December 2015).

[37] He, R., Brunskill, E., and Roy, N. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research 40* (2011), 523–570.

[38] Hellbach, S., Himstedt, M., Bahrmann, F., Riedel, M., Villmann, T., and Böhme, H.-J. Some room for glvq: Semantic labeling of occupancy grid maps. In *Advances in Self-Organizing Maps and Learning Vector Quantization*. Springer, 2014, pp. 133–143.

[39] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research 31*, 5 (2012), 647–663.

[40] Hund, A. M., and Minarik, J. L. Getting From Here to There: Spatial Anxiety, Wayfinding Strategies, Direction Type, and Wayfinding Efficiency. *Spatial Cognition & Computation 6*, 3 (2006), 179–201.

[41] Hwang, Y. K., and Ahuja, N. A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on 8*, 1 (1992), 23–32.

[42] Johnson, C., and Kuipers, B. Efficient search for correct and useful topological maps. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (2012), IEEE, pp. 5277–5282.

[43] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research 31*, 2 (2012), 216–235.

[44] Kaess, M., Ranganathan, A., and Dellaert, F. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics 24*, 6 (2008), 1365–1378.

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

[46] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research 5*, 1 (1986), 90–98.

[47] Kim, B., and Pineau, J. Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics 8*, 1 (2016), 51–66.

[48] Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research 32*, 11 (2013), 1238–1274.

[49] Koren, Y., and Borenstein, J. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on* (1991), IEEE, pp. 1398–1404.

[50] Kormushev, P., Calinon, S., and Caldwell, D. G. Robot motor skill coordination with em-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on* (2010), IEEE, pp. 3232–3237.

[51] Kuipers, B. Modeling spatial knowledge. *Cognitive Science 2*, 2 (1978), 129–153.

[52] Kuipers, B. The spatial semantic hoierarchy. *Artificial intelligence 119*, 1 (2000), 191–233.

[53] Kuipers, B., and Byun, Y.-T. A robust, qualitative method for robot spatial learning. In *AAAI* (1988), vol. 88, pp. 774–779.

[54] KURNIAWATI, H., DU, Y., HSU, D., AND LEE, W. S. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research* (2010), 0278364910386986.

[55] KURNIAWATI, H., HSU, D., AND LEE, W. S. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems* (2008), vol. 2008.

[56] LAI, K., BO, L., REN, X., AND FOX, D. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (2011), IEEE, pp. 1817–1824.

[57] LAPIERRE, L., AND SOETANTO, D. Nonlinear path-following control of an auv. *Ocean Engineering 34*, 11 (2007), 1734–1744.

[58] LATOMBE, J.-C. *Robot Motion Planning*. The Springer International Series in Engineering and Computer Science. Springer, December 1991.

[59] LAVALLE, S. M. Rapidly-exploring random trees: A new tool for path planning.

[60] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[61] LIKHACHEV, M., FERGUSON, D. I., GORDON, G. J., STENTZ, A., AND THRUN, S. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS* (2005), pp. 262–271.

[62] LOVELACE, K., HEGARTY, M., AND MONTELLO, D. Elements of Good Route Directions in Familiar and Unfamiliar Environments. In *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science*, vol. 1661. Springer Berlin / Heidelberg, 1999, pp. 751–751.

[63] LYNCH, K. *The Image of the City*. Publications of the Joint Center for Urban Studies. Technology Press, 1960.

[64] MACMAHON, M. T. *Following natural language route instructions*. PhD thesis, Austin, TX, USA, 2007.

[65] MAHADEVAN, S., AND CONNELL, J. Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence 55*, 2-3 (1992), 311–365.

[66] MALOOF, M. A. Learning when data sets are imbalanced and when costs are unequal and unknown. In *ICML-2003 workshop on learning from imbalanced data sets II* (2003), vol. 2, pp. 2–1.

[67] MEHTA, D., FERRER, G., AND OLSON, E. Fast discovery of influential outcomes for risk-aware MPDM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (May 2017).

[68] MICHON, P.-E., AND DENIS, M. When and Why Are Visual Landmarks Used in Giving Directions? In *Spatial Information Theory*, vol. 2205 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pp. 292–305.

[69] MONTEMERLO, M., THRUN, S., KOLLER, D., WEGBREIT, B., ET AL. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai* (2002), pp. 593–598.

[70] MOZOS, O. M., MIZUTANI, H., JUNG, H., KURAZUME, R., AND HASEGAWA, T. Categorization of indoor places by combining local binary pattern histograms of range and reflectance data from laser range finders. *Advanced Robotics 27*, 18 (2013), 1455–1464.

[71] MOZOS, O. M., STACHNISS, C., AND BURGARD, W. Supervised learning of places from range data using adaboost. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on* (2005), IEEE, pp. 1730–1735.

[72] MOZOS, O. M., TRIEBEL, R., JENSFELT, P., ROTTMANN, A., AND BURGARD, W. Supervised semantic labeling of places using information extracted from sensor data. *Robotics and Autonomous Systems 55*, 5 (2007), 391–402.

[73] NIEKUM, S., CHITTA, S., BARTO, A. G., MARTHI, B., AND OSENTOSKI, S. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems* (2013), vol. 9.

[74] OLSON, E. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (May 2011).

[75] OLSON, E. M3rsm: Many-to-many multi-resolution scan matching. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on* (2015), IEEE, pp. 5815–5821.

[76] OLSON, E., AND AGARWAL, P. Inference on networks of mixtures for robust robot mapping. In *Proceedings of Robotics: Science and Systems (RSS)* (Sydney, Australia, July 2012).

[77] OLSON, E., STROM, J., MORTON, R., RICHARDSON, A., RANGANATHAN, P., GOEDDEL, R., BULIC, M., CROSSMAN, J., AND MARINIER, B. Progress towards multi-robot reconnaissance and the MAGIC 2010 competition. *Journal of Field Robotics 29*, 5 (September 2012), 762–792.

[78] PAPADIMITRIOU, C. H., AND TSITSIKLIS, J. N. The complexity of markov decision processes. *Mathematics of operations research 12*, 3 (1987), 441–450.

[79] PINEAU, J., GORDON, G., THRUN, S., ET AL. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI* (2003), vol. 3, pp. 1025–1032.

[80] POMERLEAU, D. A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation 3*, 1 (1991), 88–97.

[81] PRENTICE, S., AND ROY, N. The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. In *Robotics Research*. Springer, 2011, pp. 293–305.

[82] PRESSON, C. C., AND MONTELLO, D. R. Points of reference in spatial cognition: Stalking the elusive landmark*. *British Journal of Developmental Psychology 6*, 4 (1988), 378–381.

[83] RANGANATHAN, A., AND DELLAERT, F. Online probabilistic topological mapping. *The International Journal of Robotics Research 30*, 6 (2011), 755–771.

[84] RICHTER, K.-F. A uniform handling of different landmark types in route directions. In *Spatial Information Theory*, vol. 4736. Springer Berlin / Heidelberg, 2007, pp. 373–389.

[85] RICHTER, K.-F., AND DUCKHAM, M. Simplest Instructions: Finding Easy-to-Describe Routes for Navigation. In *GIScience* (2008), pp. 274–289.

[86] RICHTER, K.-F., AND KLIPPEL, A. A Model for Context-Specific Route Directions. In *Spatial Cognition IV. Reasoning, Action, Interaction*, vol. 3343 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 58–78.

[87] SERAFIN, J., AND GRISETTI, G. Nicp: Dense normal based point cloud registration. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (2015), IEEE, pp. 742–749.

[88] SHI, L., KODAGODA, S., AND DISSANAYAKE, G. Laser range data based semantic labeling of places. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on* (2010), IEEE, pp. 5941–5946.

[89] SIMHON, S., AND DUDEK, G. A global topological map formed by local metric maps. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on* (1998), vol. 3, IEEE, pp. 1708–1714.

[90] SORDALEN, O., AND DE WIT, C. C. Exponential control law for a mobile robot: Extension to path following. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on* (1992), IEEE, pp. 2158–2163.

[91] SOUSA, P., ARAÚJO, R., AND NUNES, U. Real-time labeling of places using support vector machines. In *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on* (2007), IEEE, pp. 2022–2027.

[92] SPONG, M. W., HUTCHINSON, S., AND VIDYASAGAR, M. *Robot modeling and control*, vol. 3. Wiley New York, 2006.

[93] STENTZ, A. The focussed dˆ* algorithm for real-time replanning. In *IJCAI* (1995), vol. 95, pp. 1652–1659.

[94] STONE, P., SUTTON, R. S., AND KUHLMANN, G. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior 13*, 3 (2005), 165–188.

[95] STREETER, L. A., VITELLO, D., AND WONSIEWICZ, S. A. How to tell people where to go: comparing navigational aids. *International Journal of Man-Machine Studies 22*, 5 (1985), 549 – 562.

[96] SÜNDERHAUF, N., AND PROTZEL, P. Switchable constraints for robust pose graph slam. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (2012), IEEE, pp. 1879–1884.

[97] SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. Going deeper with convolutions. *CoRR abs/1409.4842* (2014).

[98] THORNDYKE, P. W. Distance estimation from cognitive maps. *Cognitive Psychology 13*, 4 (1981), 526 – 550.

[99] THRUN, S. Learning metric-topological maps for indoor mobile robot navigation* 1. *Artificial Intelligence 99*, 1 (1998), 21–71.

[100] THRUN, S., AND BÜCKEN, A. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence* (1996), pp. 944–951.

[101] THRUN, S., FOX, D., BURGARD, W., AND DELLAERT, F. Robust monte carlo localization for mobile robots. *Artificial Intelligence 128*, 1-2 (2001), 99–141.

[102] THRUN, S., AND MONTEMERLO, M. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. 403–430.

[103] TRIEBEL, R., SCHMIDT, R., MOZOS, Ó. M., AND BURGARD, W. Instance-based amn classification for improved object recognition in 2d and 3d laser range data. In *Proceedings of the 20th international joint conference on Artifical intelligence* (2007), Morgan Kaufmann Publishers Inc., pp. 2225–2230.

[104] TU, J. V. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology 49*, 11 (1996), 1225–1231.

[105] ULRICH, I., AND NOURBAKHSH, I. Appearance-based place recognition for topological localization. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on* (2000), vol. 2, Ieee, pp. 1023–1029.

[106] VANETTI, E. J., AND ALLEN, G. L. Communication Environmental Knowledge: The Impact of Verbal and Spatial Abilities on the Production and Comprehension of Route Directions. *Environment and Behavior 20*, 6 (November 1988), 667–682.

[107] VASUDEVAN, S., AND SIEGWART, R. Bayesian space conceptualization and place classification for semantic maps in mobile robotics. *Robotics and Autonomous Systems 56*, 6 (2008), 522–537.

[108] VISWANATHAN, P., MEGER, D., SOUTHEY, T., LITTLE, J. J., AND MACKWORTH, A. Automated spatial-semantic modeling with applications to place labeling and informed search. In *Computer and Robot Vision, 2009. CRV'09. Canadian Conference on* (2009), IEEE, pp. 284–291.

[109] WALTER, M. R., HEMACHANDRA, S., HOMBERG, B., TELLEX, S., AND TELLER, S. Learning semantic maps from natural language descriptions. Robotics: Science and Systems.

[110] ZEILER, M. D., AND FERGUS, R. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*. Springer, 2014, pp. 818–833.