Learning Multi-Modal Navigation in Unstructured Environments

Arne Jonni Suppé

CMU-RI-TR-22-22

May 4, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Martial Hebert, *Chair*
Kris Kitani
Jean Oh
Junsong Yuan, *State University of New York at Buffalo*

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Robotics.*

This thesis is dedicated to my beloved wife, who encouraged me through the long years of work, my parents, who instilled within me the curiosity that compels me to learn more, and the teachers who have guided me on the journey.
Thank you.

Abstract

A robot that operates efficiently in a team with humans in an unstructured outdoor environment must translate commands into actions from a modality intuitive to its operator. The robot must be able to perceive the world as humans do so that the actions taken by the robot reflect the nuances of natural language and human perception. Traditionally, a navigation system combines individual perception, language processing, and planning blocks that are often trained separately with different performance specifications. They communicate with restrictive interfaces to ease development (i.e., point objects with discrete attributes and a limited command language), but this also constrains the information one module can transfer to another.

The tremendous success of deep learning has revolutionized traditional lines of research in computer vision, such as object detection and scene labeling. Visual question answering, or VQA, connects state-of-the-art techniques in natural language processing with image understanding. Symbol grounding, multi-step reasoning, and comprehension of spatial relations are already elements of these systems. These elements are unified in an architecture with a single differentiable loss, eliminating the need for well-defined interfaces between modules and the simplifying assumptions that go with them.

We introduce a technique to transform a text language command and a static aerial image into a cost map suitable for planning. We build upon the FiLM VQA architecture, adapt it to generate a cost map, and combine it with a differentiable planning loss (Max Margin Planning) modified to use the Field D* planner. With this architecture, we take a step towards unifying language, perception, and planning into a single, end-to-end trainable system.

We present an extensible synthetic benchmark derived from the CLEVR dataset, which we use to study the comprehension abilities of the algorithm in the context of an unbiased environment with virtually unlimited data. We analyze the algorithm's performance on this data to understand its limitations and propose future work to address its shortcomings. We offer results on a hybrid dataset using real-world aerial imagery and synthetic commands.

Planning algorithms are often sequential with a high branching factor and do not map well to the GPUs that have catalyzed the development of deep learning in recent years. We carefully selected Field D* and Max Margin Planning to perform well on highly parallel architectures. We introduce a version of Field D* suitable for multi-GPU data-parallel training that uses the Bellman-Ford algorithm, boosting performance almost ten times compared to our CPU-optimized implementation.

The fluid interaction between humans working in a team depends upon a shared understanding of the task, the environment, and the subtleties of language. A robot operating in this context must do the same. Learning to translate commands and images into trajectories with a differentiable planning loss is one way to capture and imitate human behavior and is a small step towards seamless interaction between robots and humans.

Acknowledgments

I owe a great debt of gratitude to many people who have made this research possible. My thesis advisor, Martial Hebert, has patiently mentored and supported me through this process, giving me the freedom to explore a topic of my choosing while providing ideas and observations which helped me overcome challenges. I would also like to thank my committee members, Kris Kitani, Jean Oh, and Junsong Yuan, for their council over the years, their perspectives on how my work fits in the big-picture, and their invaluable suggestions on how I can improve my research. Thank you for all your contributions and encouragement.

I want to thank current and past members of the NavLab research group, Dave Duggins, John Kozar, Jay Gowdy, Christoph Mertz, and Aaron Steinfeld, with whom I have many memorable experiences in building and testing robotic systems.

My longtime officemate, Luis Navarro-Serment, has provided wisdom and humor in equal parts through many projects over the years.

Ben Brown introduced me to robotics, and Mark Stehlik and Chuck Thorpe advised me as an undergraduate during my first independent research project, The SCS Autonomous Buggy.

I would also like to thank the faculty, staff, and countless students at the Robotics Institute and School of Computer Science who I have had the pleasure to know. Their intellect and willingness to share knowledge are what make it such a wonderful place to work and do research.

To my family, thank you for nurturing my childhood curiosities and always buying the supplies I required for my experiments.

To my loving wife, Yangfang, your constant support and encouragement have been instrumental in completing this thesis; I could not have finished without you.

To my dear son, Martin, who amazes me with his joy in learning, Dada's "story" is complete.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A robot that operates seamlessly with a human must translate commands into the desired actions. In scenarios as disparate as disaster recovery, combat, and elder care, it can not be assumed that the operator has the time or the training to generate detailed instructions using specialized interfaces. In short, the ideal robot would follow instructions as well as any other human teammate, perhaps with text or spoken language serving as the command interface, and make appropriate assumptions about ambiguous command or perception data to execute the intended actions.

This challenging problem lies at the intersection of perception, natural language processing, and artificial intelligence. In this thesis, we plan to address just one example of this large and very significant problem: how to traverse an aerial image, given starting and ending locations and a spoken or written command. Even in this restricted context, we face challenges similar to the larger problem; both language and perception are often ambiguous and are only clarified using the large amount of implicit knowledge that underlays our everyday interactions with the world and each other.

Almost as soon as the first mobile robots were made, researchers have been looking towards natural languages as an ideal way to command them. Shakey the Robot (Nilsson [4]) used a command language that was directly translated into a first-order predicate calculus statement. It operated in an office environment specially constructed to make the perception problem as simple as possible. Shakey's world was distilled into an array of predicate calculus statements. A plan was a sequence of elementary actions that manipulated the world model into the desired state. In Shakey's case, perception was assumed to directly yield discrete symbols that could be processed with classical artificial intelligence techniques such as logical inference. Indeed, modeling perception was explicitly not a design objective of this system to simplify an already complex problem of integrating language and symbols to produce a plan.

Many early robots used sonar and lidar as primary sensors. Sonar measures the presence of an obstacle with a physical process that is well understood and straightforward to model mathematically, making it possible to directly integrate these models with algorithms like occupancy grids (Moravec

[5]), particle filter localizers (Thrun [6]) and SLAM systems (Thrun *et al.* [7]). In turn, these systems have been integrated into the intelligence and navigation architectures of robots that operate in many different environments, like CARMEL (Congdon *et al.* [8]) and Boss (Urmson *et al.* [9]).

Unlike sonar or lidar, computer vision is uniquely challenging to model because even the simplest of objects can be difficult to describe with mathematical precision even before considering the sheer volume of noisy and often ambiguous data in real-world images. Marr [10] is an example of a pioneering attempt to model human vision processing at varying levels of abstraction by combining fundamental features such as contours and blobs to match object models. Ultimately, this concept has proven intractable outside of machine vision applications and does not represent modern object detection systems, which usually haven't any explicit object models. Instead, the model is internal to a deep neural network, having been learned from many samples and requiring no human-designed models for the countless scenarios in which that object might appear.

Because of this problem, it is also much more difficult to effectively integrate the output of higher-level vision systems with navigation architectures. Our own recent experience with outdoor field robotics (Oh *et al.* [11]) supports this observation. Simplified models are necessary to condense complex visual information from an object detector (Zhu *et al.* [12]) and a semantic scene labeler (Munoz [13]) into mathematical concepts that the navigation system can manipulate. These models are carefully engineered for a particular task (such as the exemplar 3D models in Zhu *et al.* [12]) or are extremely general, such as a bounding box with a confidence metric used to define vehicles and humans. By making these simplifications, higher-level perception information can be integrated into existing navigation architectures, but at the cost of the nuance and flexibility lost when data is discarded.

Abstract models are often difficult to maintain, given the diversity of objects and scenarios in the presence of ambiguous and noisy data. Based on recent experience with modern object detectors, it is likely better to have implicit models trained by sampling realistic data than to maintain a catalog of carefully engineered models.

Modular systems are often trained on many sub-tasks with performance metrics different from the task metric. For example, a scene labeler maximizes an intersection-over-union metric over labeled regions, but many classes may be irrelevant to the robot task. The semantic scene labeler in Munoz [13] did not distinguish the importance of road pixels from sky pixels, which are irrelevant for the ground robots in Oh *et al.* [11]. An integrated system trained end-to-end could automatically learn the behaviors that best support the overall task.

The challenge of integrating perception with navigation is compounded by the problem of translating a written or spoken command into the intended plan. The command will likely reference objects in the world that must be supported with evidence gathered by perception. Even with a highly structured language such as in Oh *et al.* [11], there remains the problem of matching the symbols in the command to imperfect perception data, in addition to modeling the physical meaning of unassumingly difficult concepts such as “to the left of” or “behind.” This process is called *grounding*, and has usually been accomplished with probabilistic inference over abstract models of the world, such as in Kollar *et al.* [14]. Again, the abstraction of perception to a simple model makes the problem tractable yet sacrifices the nuances that make visual data informative.

Computer vision has recently seen incredible development. The deformable parts model (DPM) (Felzenszwalb *et al.* [15]), a leading object detector ten years ago, has been overtaken by deep learning approaches. DPM is an exemplar of a high-performing model-based computer vision technique, highly optimized for detecting structured objects such as vehicles or people. Modern object detection algorithms do not have any explicit model and rely on vast quantities of labeled data to learn an implicit model from the data itself. Without the constraint of an apriori model, these techniques can freely adapt to the data, even when it is imperfect or ambiguous.

Image understanding and natural language processing architectures share many deep learning techniques (e.g., LSTM, RNN), supporting the natural fusion of these domains into a single framework. This has already occurred with Image Captioning (Chen *et al.* [16] and others) and Visual Question Answering (Johnson *et al.* [17] and others). The most recent examples all have some form of symbol grounding, which is essential to robot navigation with a spoken or written command. Tamar *et al.* [18] study the convergence of perception and navigation via deep learning, and Anderson *et al.* [19] study the combination of perception, language, and navigation.

We choose to approach the problem of integrating perception, natural language processing, and planning from the point of view of deep learning, using techniques in visual question answering as a starting point for our architecture, suitably modified for the unique context of our problem. Different from recent research, such as Oh *et al.* [11], the grounding mechanism is not explicitly modeled. Instead, we propose to learn the desired trajectory directly from the command and map, precluding the need for interfaces between modules and allowing for the simultaneous training of language, perception, and planning elements against the same loss metric that measures the robot's performance.

1.1 Problem Statement

We are given an aerial orthographic image $\mathcal{I}_i$, for $i \in 1...N$, a natural language command Λ_{ij}, for $j \in 1...M$, a set of starting and ending points, X_{ij} and Y_{ij}, and a set trajectories μ^*_{ij}, provided by experts (Figure 1.1). Our goal is to interpret a new command Λ', image $\mathcal{I}'$, and starting and ending locations X', Y', and generate a trajectory μ' that conforms to what an expert would have drawn. We do this by employing a deep network, whose output, $F(\mathcal{I}', \Lambda', X', Y')$, is a cost map that induces the path planner to select the trajectory which meets the constraints of the intended path[1]. We assume the world is static and fully observable. Robot navigation is simplified into a more constrained task that still poses the exciting challenge of how to unify command parsing, scene understanding, and path planning into a single, fully differentiable process.

[1] We carefully note that in its original form, cost map denotes an obstacle/non-obstacle map. We adopt the convention used by the Maximum Margin Planning algorithm of Ratliff *et al.* [20]. Although modeled on an MDP, it is a structured learning algorithm, and the authors specifically refer to the output as a cost map and not a reward function. Other works, such as MaxEnt IRL (Ziebart *et al.* [21]) and its derivatives do estimate an underlying reward function of an MDP. We will refer to either as a cost map in general and use the term value function when writing specifically in the context of an MDP.

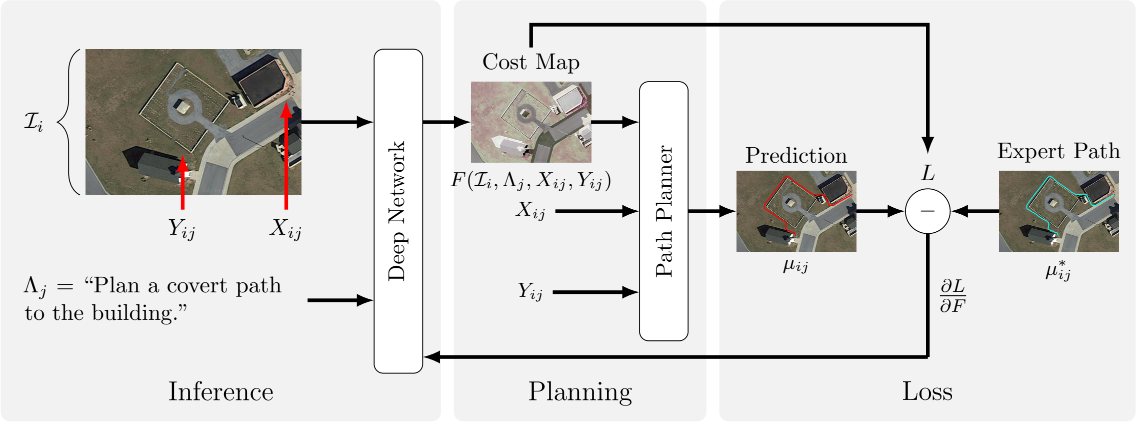


Figure 1.1: Simplified system schematic of a multi-modal planning algorithm which accepts a command (Λ_j), image ($\mathcal{I}_i$) and path endpoints (X_{ij}, Y_{ij}) and synthesizes a cost map suitable for a path planner which generates path μ_{ij}. The differentiable planning loss (L) facilitates end-to-end learning and requires no explicit abstraction of perception data or symbol grounding.

1.2 Thesis Organization

In Chapter 2, we introduce prior work on how spoken or written commands and perception are integrated to generate plans. We examine their strengths and weaknesses to motivate the idea that it is necessary and useful to build a navigation system that unifies natural language processing, image understanding, and path planning with a deep network and a single loss metric.

In Chapter 3 we introduce two related Inverse Optimal Control (IOC) techniques, Maximum Margin Planning (MMP) and Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL), that have been used to imitate trajectories. We also examine Visual Question Answering (VQA) architectures to understand how they might adapt to map text commands and static images into cost maps suitable for path planning. VQA architectures now address many of the same problems that modular language-perception-planning systems have had to solve, such as symbol grounding. As demonstrated by Hu *et al.* [22] and Lu *et al.* [23] among many others, VQA systems have different performance characteristics depending on the question posed. This is often due to limitations in the grounding strategies the underlying model can express. We select the Feature-wise Linear Modulation network (FiLM) from Perez *et al.* [24] as the basis for our algorithm because it demonstrates superior performance on the CLEVR benchmark (Johnson *et al.* [25]) without an explicit model of the reasoning process used to answer a question. This is helpful because real-world computer vision tasks often defy procedural modeling. Finally, we introduce the metrics we will use to evaluate system performance.

We begin Chapter 4 by modifying the Maximum Margin Planning loss for a trivial problem with a simple neural network to understand what is required to make it numerically stable for more complex architectures. Next, we show how FiLM, a vastly more complex deep network, can be modified for our task. We use Dijkstra's algorithm (Dijkstra [26]) as the planner for the initial work and later demonstrate how Field D* can be adapted to work with MMP to learn and produce more realistic paths.

In Chapter 6 we introduce the datasets we use to evaluate the performance of our algorithm. As we have not found a dataset suited to our problem, we have adapted two existing ones. The CLEVR dataset (Johnson *et al.* [25]) was initially designed as a benchmark for VQA systems that uses synthetic images and questions with no external bias that a careful observer may exploit. The data are generated using a script that we have re-engineered to create planning scenarios. As with CLEVR, we introduce a mechanism to ensure that our commands require correct reasoning about both the image and the command and for which default strategies are unlikely to perform well. We have also modified the DOTA/iSAID dataset (Dataset of Object deTection in Aerial images) (Ding *et al.* [27], [28], Xia *et al.* [29], and Zamir *et al.* [30]), a large dataset of annotated aerial imagery, to generate a new hybrid dataset with real objects and synthetic commands. We explore the performance of our algorithm under varying conditions and analyze its limitations to inform future directions of research.

Deep learning's success is intimately tied to massively parallel computation engines and carefully selected algorithms suited to these architectures. However, robot path planning algorithms are not generally part of the repertoire of tools that come with frameworks such as PyTorch or TensorFlow. In Chapter 5 we discuss the algorithms selected for this work and their adaptation for GPU architectures. While time-consuming, this work is also necessary. An algorithm with excessive run time is difficult to develop and test and impractical to deploy.

We conclude the thesis with Chapter 7, restating the contributions of this work but also highlighting unanswered questions and future directions for continuing research.

1.3 Contributions

The significant contributions of this thesis are as follows:

- An architecture for unifying natural language processing, image understanding, symbol grounding, and path planning into a single deep learning framework with a single evaluation metric. We evaluate the algorithm on two datasets, highlighting strengths and weaknesses and identifying directions for future work.
- A synthetic dataset, based on the CLEVR (Johnson *et al.* [25]) and DOTA (Ding *et al.* [28]) datasets, for testing the reasoning abilities of this and similar systems in an unbiased environment with virtually unlimited data. The dataset generation scripts have been re-targeted for our problem space but are extensible and may be of use to other researchers.
- Scalable and massively parallel implementations of Field D*, the Fréchet coupling metric, and Max Margin Planning that enable training and evaluation of deep networks with planning losses without CPU limitations. These tools are an essential bridge between theoretical contributions and future practical applications.

Chapter 2

Commanding Robots with Language

This chapter summarizes recent work in robotics where language and perception are combined to plan robot paths. Second, we present our practical experience with a robot tested in an outdoor environment. We do this to understand what kinds of problems these systems solve and what opportunities exist for improvement.

2.1 Prior Work

Symbol grounding is the process of relating symbols in the form of words to concrete representations in perception data first formulated in Harnad [31]. This process is essential for planning robot trajectories with language commands. Shakey (Nilsson [4]) did this by directly translating the instruction into the same predicate calculus statements that comprised the world model. Perception was vastly simplified by engineering the robot's environment to minimize clutter and false detections. These discrete and unambiguous object detections were translated into more predicate calculus symbols in the world model. There was no ambiguity in understanding the meaning of the command or the state of the world.

The real world is not nearly so ideal. Commands may be ambiguous because of information they leave out or assume as implicit or even because one person's understanding of a phrase may differ from another (Oh *et al.* [11]). Objects may be in a cluttered environment, partially obstructed, have multiple or false detections, or may have an appearance that does not exactly match the exemplar object.

One method for dealing with ambiguity and error is to develop appropriate fallback and error recovery routines. In MacMahon *et al.* [32], the authors assume that only the instruction may be ambiguous and not perception. The simulated environment in which they run human and automated trials is a maze of corridors with sparse features and a limited set of actions at each intersection.

Their core contribution is a method by which a natural language instruction is parsed into a syntax tree and then translated into a robust set of commands for the robot to follow, using hard-coded knowledge to fill in information implicit in the natural language instruction. This reduction of the world to a topological map with discrete decision points is common in literature and is appropriate for highly structured environments composed of corridors and discrete features. However, it does not scale well for open environments that lack distinctive landmarks and where one may arrive at the same goal with many solutions.

Kollar *et al.* [14] is an instructive example of how probabilistic reasoning can be used to solve the grounding problem. We expand upon this example because it is more readily summarized than more complex systems yet expresses some of the core techniques common to the others, such as codifying relational concepts from language (e.g. “left of” or “right of”) as probabilistic equations.

In Kollar *et al.* [14], the authors show that it is possible to automatically translate more than ninety percent of human-generated instructions used to describe paths in a real office environment using a hierarchical formalism called spatial description clauses (SDC), allowing for natural language communication with only occasional parsing errors. Each command is parsed into a tree of nested SDCs using a CRF to label words as objects, verbs, landmarks, or spatial relationships grouped into a tree that reflects the hierarchical structure of the command. This structure is later grounded to generate instructions that the robot can execute.

With slightly more formalism, given a path P, a sequence of SDCs S, and detected objects O, the path that best satisfies the sequence of relations specified in the SDCs is:

$$P^* = \underset{P}{\operatorname{argmax}}\, p(P, S|O) = \underset{P}{\operatorname{argmax}}\, p(S|P, O)p(P|O) \tag{2.1}$$

Perception is not modeled in this process as the objects, O, are given. In addition, the algorithm returns the most probable path to the exclusion of others. The authors decompose the path into discrete viewpoints v_i in a topological map, with each SDC inducing a change in viewpoint with the following approximation:

$$p(P, S|O) \approx p(sdc_1, ...sdc_M|v_1...v_{M+1}, O)p(v_1...v_{M+1}, O) \tag{2.2}$$

$$\approx \left[\prod_{i=1}^{M} p(sdc_i|v_i, v_{i+1}, O)\right] \left[\prod_{i=1}^{M} p(v_{i+1}|v_i, ...v_1)\right] p(v_1) \tag{2.3}$$

Finally, each SDC is decomposed into the underlying grammar model by expanding it into an object (f), verb (a), landmark (l), and spatial relationship (s) (Equation (2.5)).

$$p(sdc_i|v_i, v_{i+1}, O) = p(\underbrace{f_i, a_i, s_i, l_i}_{sdc_i}|v_i, v_{i+1}, O) \tag{2.4}$$

$$\approx \underbrace{p(f_i|v_i, v_{i+1}, o_1...o_K)}_{\text{object}} \underbrace{p(a_i|v_i, v_{i+1})}_{\text{action}} \underbrace{p(s_i|l_i, v_i, v_{i+1}, o_1...o_K)}_{\text{spatial relation}} \underbrace{p(l_i|v_i, v_{i+1}, o_1...o_K)}_{\text{landmark}} \tag{2.5}$$

The action term in Equation (2.5) is a hard-coded model that relates terms such as "left" or "straight" to the physical motion required to move from v_i to v_{i+1}. The landmark and object terms are computed using image co-occurrence statistics gathered from Flickr. That is, given the objects o_k that are visible at v_i, $p(l_i|...)$ is the most likely pairwise co-occurrence between o_k and l_i. Finally, the spatial relation term encodes phrases like "through" or "to the left of." This is learned from a distribution of examples drawn by humans. Perception is not modeled in any meaningful way. The robot's world is segmented into a topological map, where again, there are discrete actions and viewpoints.

The key difference from MacMahon *et al.* [32], however, is that the grounding model is probabilistic. The model's goal is to jointly predict the most likely path between viewpoints in the topological map with the command, given landmark detections. The authors use hand-engineered features to compute the probability of how landmarks and spatial relations comport with the command. For example, spatial relations for 11 prepositions were learned from hand-drawn examples from humans.

This method, however, has no explicit model for noisy object detections. The probabilistic formulation for grounding makes the model more flexible. Still, as it selects the path that maximizes the joint probability of the path and command, it cannot recover from errors. It avoids the need to create an explicit model of hard to define human expressions by learning the meaning of spatial relations from human examples. However, it assumes the meaning of the spatial relation is independent of the object, its orientation, or even the rest of the command. Further, specific elements of the command language map directly to the hand-engineered features. Real language is not so regular or so easily decomposed. Finally, the model reasons about paths in the context of a topological map, making it easier to perform inference over a discrete set of positions and orientations, but which may not be appropriate for outdoor and unstructured environments.

Other systems solve these problems with two techniques. First, the command language is usually restricted to a limited set of nouns, actions, and modifiers that take a regular form that is easily parsed into a syntactic tree (Boularias *et al.* [33], Guadarrama *et al.* [34], Hemachandra *et al.* [35], and others). By limiting the selection of nouns, we can train detectors for well-defined object classes that map directly to them.

The second strategy condenses discrete objects in perception data to a form suitable for Bayesian inference on a graphical model. The graph itself is a relational representation of how the various symbols and the command's structure comport to perception features. By limiting the types of modifiers (e.g. "left of", "above", etc.), we can develop scoring mechanisms to determine how well a hypothesized configuration of the graph conforms to the statement.

Table 2.1 lists several other navigation systems that we have investigated. For example, Gupta *et al.* [36] show that its possible to learn cost maps for each image that a robot sees as a travels through an office environment. Anderson *et al.* [19] use reinforcement learning to follow a path through a home environment that has a topological map. None of the literature we have seen, to this point, also integrate language and the paths are usually constrained by nature of the map (i.e. topological) or because the plan is very short term. While each have their strengths, none of them cover all the properties that we are interested in. More recently, Chen *et al.* [37] is a vision language

Table 2.1: Comparison of some of the properties of robot navigation systems.

	Kollar *et al.* [41]	Anderson *et al.* [19]	Tamar *et al.* [18]	Misra *et al.* [42]	Wulfmeier *et al.* [43]
	Probabilistic	LSTM	Value Iteration Net	Deep RL	Deep IRL
Map Type	Topological	Topological	Metric	Metric	Metric
Deep Features	No	Yes	Yes	Yes	Yes
Structured Prediction	No	No	Yes	No	Yes
Grounding / Attention	Yes	Yes	No	Yes	No
Language	Yes	Yes	No	Yes	No

navigation system that solves a similar problem to Anderson *et al.* [19] using a graph neural network on a topological map. Wang *et al.* [38] finds traversable paths for a vehicle in a problem similar to Wulfmeier *et al.* [39], but without language. Song [40] is an example of deep inverse reinforcement learning for planning in unstructured environments using single words to condition the map. In summary, we choose to explore the problem of combining language and vision for navigation on a map for use in unstructured environments, which is less developed than other related areas.

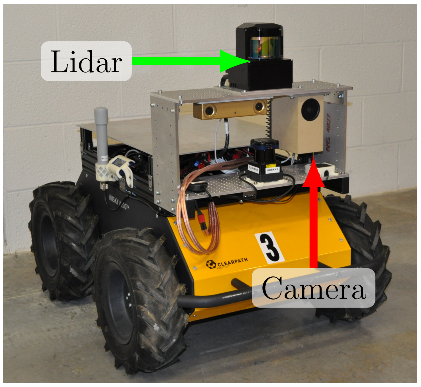
2.2 An Example Architecture

We have experience with a field robot that uses a text-based command language and perception to plan paths without any apriori maps as part of the Robotics Collaborative Technology Alliance (RCTA) program sponsored by the United States Army Research Laboratories (ARL). This work has been published in Oh *et al.* [1], Oh *et al.* [11], Oh *et al.* [44], Suppé *et al.* [45], and Oh *et al.* [46] and others, with the complete navigation system independently evaluated in Lennon *et al.* [47]. The robot operates in an outdoor environment where discrete landmarks are often sparse, unlike some prior work that used topological maps suited for indoor environments.

The robot platform is based on a Clearpath$^{\mathrm{TM}}$ Husky (Figure 2.1) equipped with a high dynamic range camera with a 120-degree horizontal field of view and a scanning lidar with a 360-degree field of view and an 80-meter range. The lidar was used to detect buildings and walls and position labeled image pixels in 3D.

Our image classifier is the Hierarchical Inference Machine (HIM, Munoz [13]). This pixel labeler segments the image into superpixels and then labels the superpixels using a forest of decision trees to classify the contents of each superpixel based on features such as dense SIFT (Lowe [49]), Local Binary Patterns (LBP, Ojala *et al.* [50]), and texture (Shotton *et al.* [51]). Neighboring superpixels are grouped into a hierarchical structure with regional features so that it uses global as well as local information in the final labeling. The image classifier works in parallel with an object detector (Zhu *et al.* [12]) and a person detector (Yang *et al.* [48]).

Figure 2.3 shows the construction of the command language called the tactical behavior specification (TBS) used in our experiments. It is a structured language expressed in Backus-Naur



Years	2013-present
Wheelbase	≈1.5m
Weight	50kgs
Camera	GDRS Adonis HDR
Resolution	1280x768
HFOV	120°
Lidar	GDRS XR LADAR
Range	≈80m
HFOV	120°
VFOV	45°
CPU	i7-3615QM@2.3GHz × 4
Object Detector	Active DPM (Zhu *et al.* [12])
Person Detector	(Yang *et al.* [48])
Image Classifier	HIM (Munoz [13])
Sec / Frame	≈ 2.5
Resolution	640x384
Features	SIFT, LAB, Texture

Figure 2.1: Technical details for Husky robots used in Oh *et al.* [1].

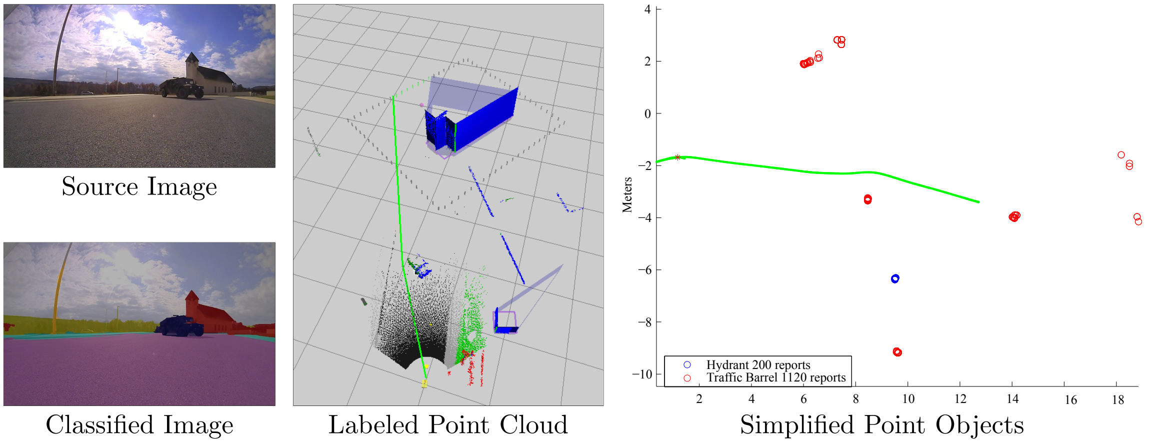
form. The named objects map directly to scene labeler and object detector categories, so there is no ambiguity in their meaning. The TBS encodes several spatial relationships, such as "left of" or "behind" and two modal constraints: "covertly" and "quickly." This affects the kind of path the robot may take in a hostile or friendly environment.

The grounding process consists of two steps. First, nouns in the TBS are mapped to observations in the world. This is accomplished with a grounding graph, detailed in Duvallet [52]. As with other navigation systems, these actions are grounded against simplified representations of the perception system (Figure 2.2), with points representing most small objects and cylinders, planes, and boxes for more voluminous ones, discarding any more detailed information beyond object class and a confidence metric. Second, the meanings of phrases such as "behind" are expressed using a cost map that is learned using Maximum Margin Planning (Ratliff *et al.* [20]), as detailed in Boularias *et al.* [33] and Boularias *et al.* [53].

Our field experiments are reported in Oh *et al.* [1] and independently evaluated in Lennon *et al.* [47]. The test environment was a simulated town situated on a military base designed to train soldiers. Of 46 runs with 30 unique TBS commands, 35 were considered successful. However, the interface between perception and planning was a common problem during the tests. For example, misperception on the part of the semantic classifier often populated the robot's map with phantom objects, causing the robot to ground the command against objects that did not exist. Moreover, the image classifier was trained to perform well on a metric that is probably not the correct one to achieve the best performance for the robot's overall task.

In the RCTA architecture and many others, information flows in one direction, from perception to grounding/intelligence processes. The perception process is essentially autonomous from the rest of the robot, making development and testing much more manageable and presenting a few problems.

First, the perception process is not trained with a loss function relevant to the task set. For


Source Image

Classified Image

Labeled Point Cloud

Simplified Point Objects

Figure 2.2: Example of perception data flow in RCTA system for semantic scene labeler. Labeled images are fused with 3D lidar to produce a labeled point cloud that is abstracted to point objects with attributes, discarding a large amount of scene data.

example, the HIM semantic classifier in Munoz [13] was designed to be competitive on the Stanford Background Dataset (Gould *et al.* [54]) and the MSRC-21 dataset (Shotton *et al.* [51]), which have many labels that are not relevant. Performance is measured by the number of correctly labeled pixels in total or averaged across classes. For a ground robot, each pixel in a scene will not have the same importance due to perspective effects, and some pixels may have little relevance to the task at all (e.g., sky pixels).

It is important to match the performance of the classifier to the robot's task, but it is not apparent how to do that. For example, a mislabeled grass region may not be as important as a mislabeled goal object. There may be multiple detection systems, each with its own characteristics and training criterion and no clear way to integrate the information into a single world model.

Each module is trained on individual corpora of task-relevant images independent of each other and independent of the robot's test scenario. In part, this is because each module was developed separately from the others to simplify collaboration, but a scene labeler may be helpful to an object detector by providing information about the image context. For example, sky pixels are unlikely to contain true-positive observations of humans but may contain birds or airplanes.

The interfaces between one module and the next are similarly difficult to define. In our case, labeled pixels are mapped to 3D points with a calibrated lidar. These point clouds were then condensed into simplified objects in the robot's local map. These models may not be appropriate for all kinds of objects and tasks, and each model must be engineered by hand. For example, color may be important for one object in a particular task, and relative orientation may be important for another object and task.

The task does not affect how the perception system works. All image regions have equal importance, even if the region is irrelevant to the command in question. This wastes computational power but also invites unnecessary errors. For example, a system planning a path through an office

```
             ⟨tbs⟩  ⊨  ⟨action⟩ [⟨direct-obj⟩] [⟨mode⟩] ⟨goal⟩ [⟨goal-constraint⟩]
          ⟨action⟩  ⊨  ⟨navigate⟩ | ⟨search⟩ | ⟨observe⟩
      ⟨direct-obj⟩  ⊨  ⟨named-obj⟩
       ⟨named-obj⟩  ⊨  ⟨named-object1⟩ | ⟨named-obj2⟩
      ⟨named-obj1⟩  ⊨  Robot | Building | Wall | Door | Grass | Asphalt | Concrete
      ⟨named-obj2⟩  ⊨  Person | TrafficBarrel | Car | GasPump | FireHydrant
            ⟨mode⟩  ⊨  ⟨simple-mode⟩{⟨path-constraint⟩}
     ⟨simple-mode⟩  ⊨  quickly | covertly
 ⟨path-constraint⟩  ⊨  ⟨constraint-list⟩
 ⟨goal-constraint⟩  ⊨  ⟨constraint-list⟩
 ⟨constraint-list⟩  ⊨  ⟨constraint⟩ | ⟨constraint⟩ { ⟨operator⟩ ⟨constraint⟩ }
      ⟨constraint⟩  ⊨  [not] ⟨spatial-relation⟩⟨landmark⟩ | [not] (⟨constraint-list⟩)
⟨spatial-relation⟩  ⊨  left | right | behind | front | around | near | away
        ⟨landmark⟩  ⊨  ⟨named-object⟩
        ⟨operator⟩  ⊨  ⟨and⟩ | ⟨or⟩
            ⟨goal⟩  ⊨  to | ⟨spatial-relation⟩ ⟨named-obj⟩
```

Figure 2.3: Tactical Behavior Specification (TBS). Figure adapted from Boularias *et al.* [33].

environment may choose to look for chairs in specific regions of an image and can safely dispense with looking for vehicles unless presented with extraordinary data. The algorithm should have some expectation as to what can be found in an environment given a task.

2.3 Improving Existing Architectures

Based on our own experience and our survey of other state-of-the-art navigation systems similar, we have identified three potential research directions to improve the state-of-the-art.

- **Training with a single loss metric:** The complete navigation system should be trained with a single loss metric. Individual modules developed in isolation may not perform optimally when combined into a navigation system.
- **Data-defined interfaces:** The interface between perception, language processing, and planning must be more flexible and expressive. Ideally, communications should be multi-lateral so that information from one module can be used by others.
- **Task-specific planning/perception:** The perception/planning process is complex but needs only support the behaviors required to achieve optimal performance for the overall task. A navigation system with a state-of-the-art object detector for ImageNet (Deng *et al.* [55]) may not be as good as a custom detector architecture for a few task-relevant objects. Further, object detections by themselves may have lower performance than mid-level features, which convey more information but in a less accessible form.

2.4 Summary

Weaving together the many disparate modules that compose a robotic navigation system is not easy. The system architecture helps compartmentalize development and complexity and is essential to collaborative work. However, with increasing computational and data resources combined with end-to-end learning, we can fuse previously separate tasks, eliminating the need for discrete interfaces and optimizing the system for peak performance under a single metric, addressing the points mentioned earlier.

As an example of the rapid progress toward data-driven monolithic architectures, consider the Discrete Parts Model (DPM, Felzenszwalb *et al.* [15]), which was a state of the art object detector only a few years ago. The DPM contains a flexible and abstract framework for modeling the HOG features in different regions of an object as the object deforms or changes position and orientation. In a few short years, DPM has been replaced by many different neural network architectures that are essentially model-free, with no explicit features or model for object appearance except for what is learned directly from the training data and internalized in the network weights.

Integrating path planning, vision, and language processing into a single deep learning framework is consistent with this trend and neatly solves several problems we have identified in existing navigation approaches. Fortunately, the fusion of language parsing and image understanding is already an area of active research from which we can draw inspiration. In Chapter 3 we discuss candidate deep networks as well as differentiable planning losses that we can combine to create a novel navigation architecture.

Chapter 3

Background & Theory

This chapter introduces the relevant theory and techniques to construct a navigation system for planning robot trajectories through static aerial images using naturalistic text commands. We present deep networks that could be modified to generate cost maps for planning, the loss function used for training, and appropriate metrics to measure performance.

The primary goal of this work is to fuse perception, natural language processing, and planning with a single differentiable loss function. In Section 3.2 we introduce Inverse Optimal Control techniques for imitation learning and discuss the reasons for selecting Maximum Margin Planning (MMP) as the loss function we use for our experiments. In Section 3.3, we evaluate several Visual Question Answering (VQA) architectures to motivate the argument that they can adapt to serve as the foundation for our planning system. We explore a few networks in detail and select Feature-wise Linear Modulation (FiLM, Perez *et al.* [24]) as the base architecture for our work, primarily because it is a successful model-free approach to VQA (Section 3.3.3). Finally, we introduce the Hausdorff and Fréchet distance metrics in Section 3.4, two standard techniques for measuring the conformity of a path to a reference trajectory.

3.1 Markov Decision Processes for Robot Planning

Markov decision processes (MDP) have long been used in robot planning. In brief, an agent (robot) is said to be in state s_i with a reward $R(s_i)$. The robot may transition to state s_j with probability p_{ij}. This transition is independent of prior history. The graph of interconnected states may be sparse, as in a topological planner, or regular and metric, as is often the case for field robots path planners. Finding the policy, μ, that minimizes the expected cost of transitioning between two states is a common task. This policy is stochastic for cases where $p_{ij} \neq 1$, or deterministic when $p_{ij} \in \{0, 1\}$.

A robot operating in an office environment, for example, may have no difficulty traversing between rooms. A sparsely connected graph representing natural decision points may be an appropriate model, with a local planner spanning the gap between nodes. In contrast, a robot operating outside must contend with irregularly shaped regions with variable traversability, obstacles, and poorly defined decision points. A rasterized map with connections to a limited area around each state may be more suitable. In both cases, stochastic transitions capture uncertainty in the agent or the environment.

An MDP with a sparse topological map can be adapted to model a rasterized map. However, the algorithms we use to *efficiently* solve problems modeled by one graph type are not the same for the other. The number of distinct paths connecting two points in a topological map may be large, but it is still reasonable to compute all policies connecting two states for many practical problems. This is often not the case for maps because they contain many more states, not all of which have equal importance. In this work, we constrain our model to maps with regular, square cells and connections between the 8-connected neighbors of each cell.

The Bellman equation (Equation (3.1)) computes the value, $V(s)$, for a policy, π, that has arrived at state s. The equation is recursively defined by virtue of the Markovian property, and prior state transitions do not affect future decisions. $P_{s\pi(s)}$ is a state transition probability matrix used for stochastic models. γ discounts the value of past states to keep the sequence bounded for problems with an infinite time horizon.

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s')V^{\pi(s')} \tag{3.1}$$

The related function $Q(s, a)$ (Equation (3.2)) is also recursively defined and computes the value of an action a at state s. P_{sa} is a state action probability matrix.

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s'} P_{sa}(s')V^{\pi(s')} \tag{3.2}$$

$\pi(s)$ is the optimal decision at state s according to optimal policy π. This optimal policy is defined for $V^{\pi}(s)$ and $Q^{\pi}(s, a)$ for all states $s \in S$ as:

$$\pi(s) \in \operatorname*{argmax}_{s \in S} V^{\pi}(s) \tag{3.3}$$

$$\pi(s) \in \operatorname*{argmax}_{a \in A, s \in S} Q^{\pi}(s,a) \tag{3.4}$$

An enormous number of path planning problems have been modeled as MDPs with an equally large number of techniques for finding optimal policies (Figure 3.1). The reward function $R(s)$ is unknown in the problem studied here, as human behavior is often difficult to capture in an abstract model.

Methods such as value iteration, policy iteration, and Q-learning require knowledge of the cost or reward for a state. For example, Mnih *et al.* [56] learn to play video games by using the score to determine high-value states. A robot manipulation strategy may use the end-effector proximity to a target as a similar metric to signify desirable states.

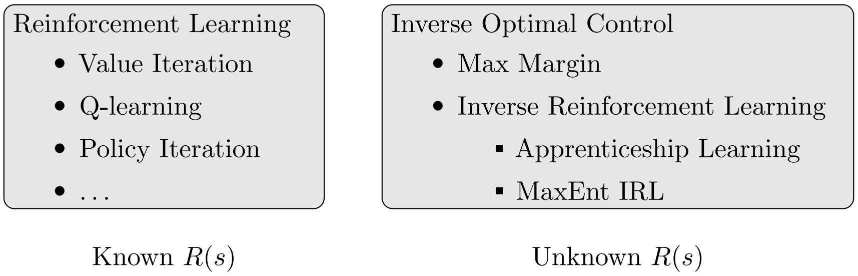


Figure 3.1: Relational diagram of various MDP methods.

Neither technique is well-suited to this problem. We envision a system that produces trajectories similar to those a human would generate, given the same command and image. As such, the system must interpret the syntax and underlying meaning of the command and image features, using a cost function identical to the human. For example, one could make states behind an object cheaper for a command that specifies a target towards the rear of an object. However, given the exponential combination of even simple geometric constraint clauses, this cost function would be difficult to code.

We pose our problem as learning a mapping from image and text inputs to path outputs by imitating expert policies which we model as an Inverse Optimal Control (IOC) problem.

3.2 Learning from Example with Inverse Optimal Control

Abbeel *et al.* [57] establish that the feature expectation (Equation (3.5)) is sufficient to determine the discounted rewards for any policy and that the goal of the IRL algorithm is to match the observed feature expectation. Equation (3.5) is the expected reward for the policy π with discount γ and a reward function $R(s_t)$.

$$E_{s_o \sim D}[V^\pi(s_0)] = E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] \tag{3.5}$$

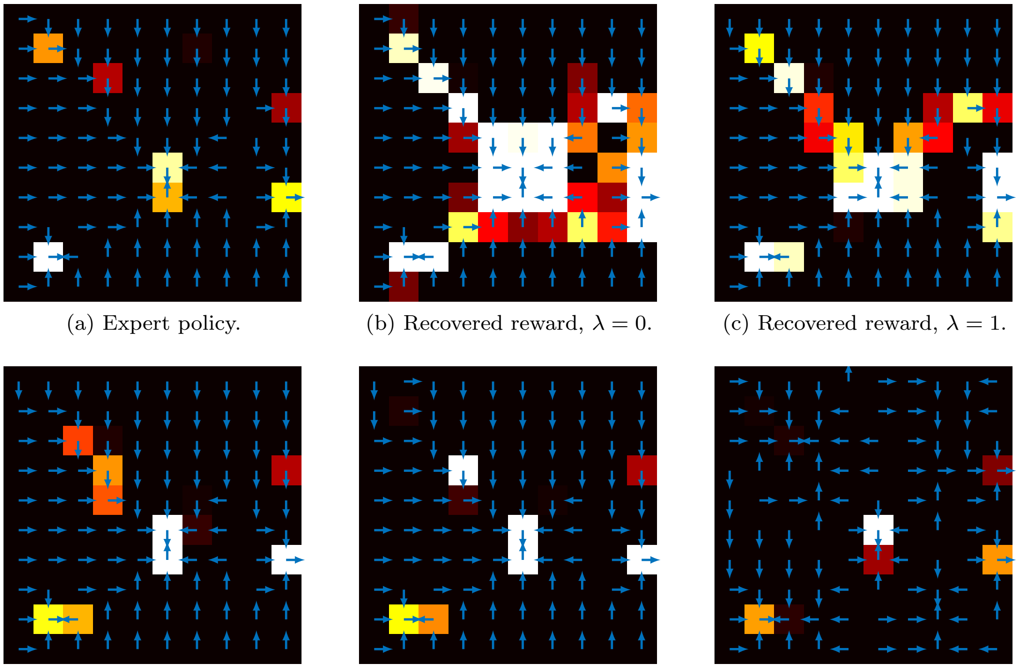
$$= E[\sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t)|\pi] \tag{3.6}$$

$$= w \cdot \underbrace{E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]}_{\text{feature expectation}} \tag{3.7}$$

Both Max Margin Planning (MMP, Ratliff [58]) and Maximum-Entropy Inverse Reinforcement Learning (MaxEnt IRL, Ziebart [59]) model the cost or reward, respectively, as a $w^T\phi(s_t)$, where $\phi(s_t)$ is a feature vector at state s_t and w is a learned set of weights (Equation (3.6)). This formulation, presented in Abbeel *et al.* [57], is particularly useful because it means that the weights w are separable from the feature expectation (Equation (3.7)). The linear separation of cost/reward and policy is a key insight of this work.

The features are often carefully engineered to suit the problem. In Ratliff *et al.* [20], the features are a vectorized representation of the source image, blurred at different scales and with a quantized colorspace. In Ziebart *et al.* [60], these features are also a blurred representation of an obstacle map for an office environment. By using deep features instead of engineered features, we are no longer constrained by the designer's skill to select features appropriate for the problem. This may make it easier to model more complicated interactions between the environment and the agent.

Ng *et al.* [61] note that this is an under-constrained problem with sparse rewards and few expert examples. Abbeel *et al.* [57] solve this with a constrained linear program. Figure 3.2 is an example of reward maps recovered from the policy (blue arrows). There are many solutions to the problem. MaxEnt IRL and MMP solve the ill-posed problem by using the principle of maximum entropy and max-margin structured learning, respectively.



(a) Expert policy.

(b) Recovered reward, $\lambda = 0$.

(c) Recovered reward, $\lambda = 1$.

(d) Recovered reward, $\lambda = 2$.

(e) Recovered reward, $\lambda = 3$.

(f) Recovered policy, $\lambda = 7$.

Figure 3.2: Linear Programming method to recover grid world cost map. Large λ will degrade result. (Bottom right)

3.2.1 Max-Margin Planning

Max-Margin Planning (MMP), introduced by Ratliff *et al.* [20], is a form of Inverse Optimal Control where the objective is to imitate a state/action visitation count, μ_i, demonstrated by an expert for some input x_i, with corresponding features $\phi(x_i)$. The variable μ is a $|\mathcal{S}||\mathcal{A}| \times 1$ vector of state/actions visited by a policy and is generally sparse. The features, $\phi(x_i)$, are a $d \times |\mathcal{S}||\mathcal{A}|$ matrix. The weights, w, are a $d \times 1$ vector that combines the features into a cost for each state/action. The algorithm is deterministic and the cost for a path is succinctly expressed as $w^T\phi(x_i)\mu_i$.

The Max-Margin Planning algorithm solves for w by optimizing over the structured loss constraint in Equation (3.8). This formulation assumes that the cost of a policy is linear with respect to its state/action visitation counts. As with Abbeel *et al.* [57], the solution to this problem is ill-defined. The authors solve this by introducing a max-margin constraint.

$$w^T\phi(x_i)\mu_i \leq \min_{\mu \in \mathcal{G}_i} \{ \left(w^T\phi(x_i) + \ell(\mu_i)^T \right) \mu \} \tag{3.8}$$

The loss function $\ell(\mu_i)$ is chosen to suit the problem. For a map, this could be a function of the distance transform from the expert path. Paths that follow the expert path should have no loss, while those that deviate should be penalized based on how far they go astray. The constraint states that the optimal policy must always have a cost less than or equal to any other policy $\mu \in \mathcal{G}_i$ between the same starting and ending states.

Ratliff *et al.* [20] show how to efficiently find a solution for w as a constrained optimization problem in Equations (3.9) and (3.10).

$$\max_{w, \zeta_i \in \mathbb{R}_+} \frac{1}{N} \sum_i \zeta_i^q + \frac{\lambda}{2}\|w\|^2 \tag{3.9}$$

$$\text{s.t.} \quad \forall i \quad w^T\phi(x_i)\mu_i + \zeta_i \leq \min_{\mu \in \mathcal{G}_i} \{ \left(w^T\phi(x_i) + \ell(\mu_i) \right) \mu \} \tag{3.10}$$

Since $\{\mu, \mu_i, \ell(\mu_i)\} \geq 0$ and $\ell(\mu_i)\mu_i = 0$, then $\zeta_i \geq 0$, and there is no need to enforce this constraint explicitly.

$$\zeta_i = \min_{\mu \in \mathcal{G}_i} \{ \phi(x_i)\mu + \ell(\mu_i) \} - \phi(x_i)\mu_i \tag{3.11}$$

Substituting ζ_i into the objective yields Equation (3.12), which is the max-margin planning objective.

$$L(\mathcal{I}_i, \mu_i, \theta) = \frac{1}{N} \sum_i \min_{\mu \in \mathcal{G}_i} \{ \phi(x_i)\mu + \ell_i\mu \} - \phi(x_i)\mu_i + \frac{\lambda}{2}\|\theta\|^2 \tag{3.12}$$

Since $\phi(x_i)$ is a cost map, there is an additional requirement that $\phi(x_i) > 0$ to avoid infinite loops. This is also a prerequisite to solve for μ using Dijkstra/A*. In our model, $\phi(x_i)$ is replaced with a deep network $\mathcal{F}(\mathcal{I}_i, \Lambda_i, \theta)$ which receives image $\mathcal{I}_i$ and language input Λ_i and is not likely to be convex.

$$L = \min_w \left(\frac{1}{N} \sum_i \frac{\beta_i}{q} \left\| w^T\phi(x_i)\mu_i - \min_{\mu \in \mathcal{G}_i} \left(w^T\phi(x_i) + \ell(\mu_i)^T \right) \mu \right\|^q + \frac{\lambda}{2}\|\theta\|^2 \right) \tag{3.13}$$

The trajectories through the state space are normalized by path length with the parameter β_i so that short sequences and long sequences are weighted equally. q is either 1 or 2, and λ is a standard weight decay.

Ratliff *et al.* [20] have demonstrated this technique's ability to imitate trajectories through an aerial image. Additionally, MMP was used by Duvallet [52] to translate command phrases, such as "left of", into robot trajectories by imitating expert examples.

3.2.2 Max-Entropy Inverse Reinforcement Learning

Inverse Reinforcement Learning is a form of Inverse Optimal Control that seeks to recreate the reward function that explains an agent's demonstrated state/action sequence. This is different from MMP which merely attempts to reproduce the agent's behavior. Since the IRL problem is well known to be ill-posed, Maximum Entropy IRL (Ziebart *et al.* [21]) utilizes the maximum entropy principle (Jaynes [62]) to learn the reward function that reproduces the expected feature counts with no more commitment to any one piece of evidence over another. This concept has been demonstrated in Ziebart *et al.* [21] to accurately predict a taxi driver's route preferences as they traverse a topological map and to predict a pedestrian's route preference through a metric grid map (Lee *et al.* [63], Ziebart *et al.* [60], and others).

As with MMP, the original work assumes the reward for any state/action is a linear combination of features engineered to suit the task. For example, the features in Ziebart *et al.* [21] include road type, speed, number of lanes, and a discrete set of maneuvers to transition from one path segment to the next (hard right, straight, etc.). The basic features for each state in Ziebart *et al.* [60] are an obstacle indicator and 4 Gaussian blurs of an occupancy map. More recent work, such as Huang *et al.* [64], uses an 819-D HOG as the basic feature.

Since the MaxEnt IRL loss function is differentiable and amenable to gradient descent optimization, Wulfmeier *et al.* [39] recently proposed a combination of Deep Learning with a MaxEnt Inverse Reinforcement loss function to learn features and the reward function simultaneously. Finn *et al.* [65] introduce a method called Guided Cost Learning, which is a sample-based MaxEnt IRL algorithm more suited for models with very large and continuous state sets, such as the configuration space of a robot arm.

MaxEnt IRL has two advantages over Max-Margin Planning. First, the MMP model assumes that there is one best solution for the structured loss. If there are multiple ways to go between two endpoints with equal cost, the result will be unpredictable, as noted in Ziebart *et al.* [60]. For example, in Figure 4.2d, if both the ground truth path and the computed path are correct, the gradient should be zero everywhere. Instead, the optimizer will receive antagonistic corrections. Further, if the features are ambiguous, MMP will not converge to a solution. Second, when planning with uncertainty, it is better to have a distribution of possible policies from which to draw alternatives.

The cost of this flexibility is often increased training time. Wulfmeier *et al.* [66] note that the computation of the expected state visitation is a bottleneck, although they do not explicitly state the cause. The map is just 100 × 100 pixels in their experiment. To capture enough visual detail, we feel that larger aerial images are necessary. In Finn *et al.* [65], the authors use adaptive sampling to estimate the partition function more efficiently. However, this technique has not yet been applied directly to computer vision.

We have chosen to first examine MMP as a differentiable loss function for our network because the minimizer is efficiently retrieved using A* at each time step, with the developmental loss function running on a CPU. As we will expand in Section 5.3, the shortest path computation can be one of the larger computational bottlenecks, stalling the GPU while the loss function is updated on the CPU. This is different from traditional deep learning computer vision applications where the GPU is the primary processor. The inherently sequential nature of the Single Source Shortest Path (SSSP) algorithm does not map well to the GPU's processing model. This has compelled us to move MMP to the GPU and switch to the Bellman-Ford-Moore algorithm, which ultimately has similar runtime to Value Iteration, which is at the core of MaxEnt IRL. MaxEnt IRL is therefore an appealing algorithm for future work.

3.3 Deep Networks for Cost Map Generation

Visual Question Answering is the problem of taking an image and a free-form, open-ended question and generating an answer to that question, often in a single word. It has been likened to the problem of solving artificial intelligence, in general.

As a motivation for our argument that VQA architectures can be used for navigation, consider Transparency by Design Networks (TBD) (Mascharka *et al.* [67]). This network is designed to generate meaningful attention maps after each layer of the recurrent network as a window into what a VQA architecture is thinking. It uses the ground truth algorithms from a procedurally generated synthetic dataset, CLEVR (Johnson *et al.* [25]), to train a network that builds a customized modular recurrent network to solve each query. The examples in Figures 3.3 and 3.4 demonstrate that the algorithm learns to parse language, ground symbols, and identify regions of space in relation to objects. These are necessary skills also required for our navigation task. However, this particular architecture has a strong model of the reasoning process, which makes it unsuitable for our work.

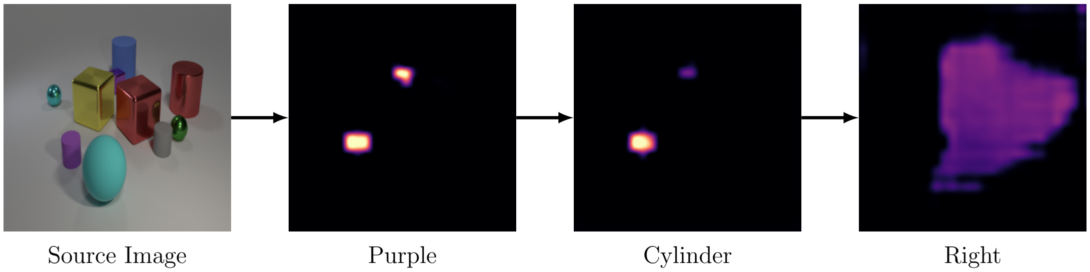


Figure 3.3: Operator sequence: Right of purple cylinder. Reproduced from Mascharka *et al.* [67].

Table 3.1 lists a few of the architectures that we studied which were high-performing at the commencement of this work. It is almost certain current VQA methods are a better choice for solving the general navigation problem. However, as a pathfinder, our primary selection criteria were that a notional base architecture has reasonable performance, training time, model size, and

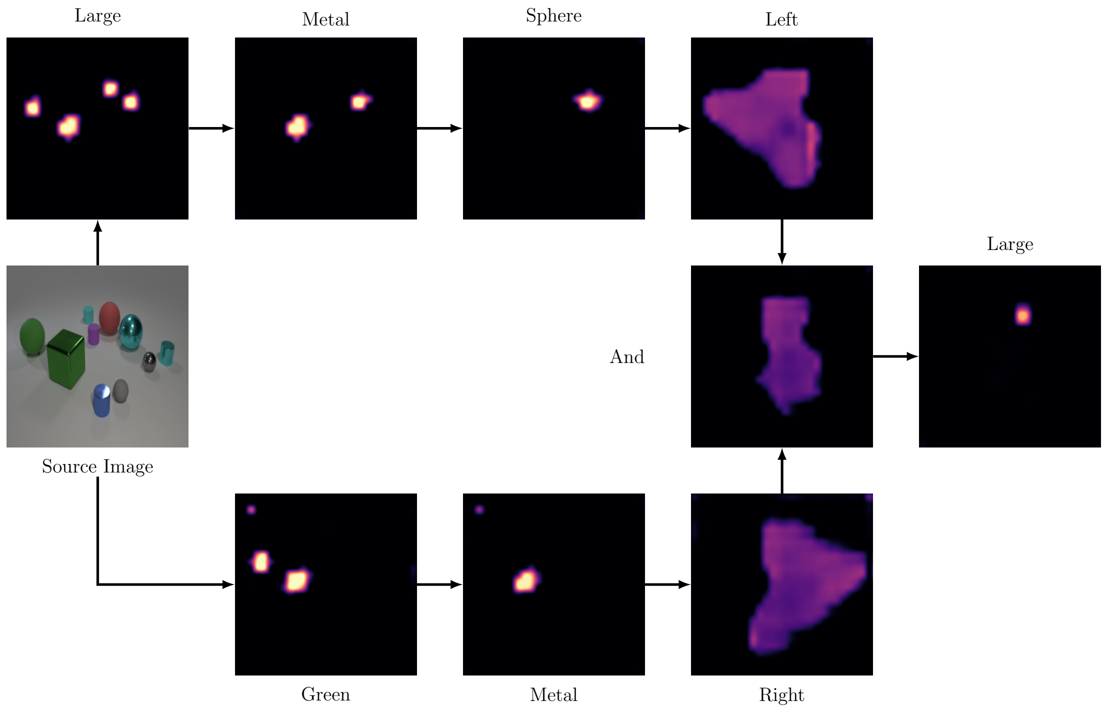


Figure 3.4: Question: What color is the big object that is left of the large metal sphere and right of the green metal thing? Reproduced from Mascharka *et al.* [67].

no internal task-specific model that would make it difficult to port to navigation-related problems. The architectures in Table 3.1 can be divided into groups that were designed for CLEVR (Johnson *et al.* [25]) or the Visual Genome (Krishna *et al.* [68]) datasets. There are major differences between techniques designed for one dataset or another.

Name	Dataset	Platform
Pythia [69]	VG	1st place VQA Challenge 2018
Multi-modal Factorized High-order Pooling (MFH) [70], [71]	VG	2nd place VQA Challenge 2018
Bi-Linear Attention Net (BAN) [72]	VG	3rd place VQA Challenge 2018
Dense CoAttention Net (DCN) [73]	VG	6th place VQA Challenge 2018
Neural-Symbolic [74]	CLEVR	
Differentiable Dynamic Reasoning (DDR) [75]	CLEVR	
Compositional Attention Networks (MAC) [76]	CLEVR	
Transparency by Design (TBD) [67]	CLEVR	

Table 3.1: Survey of leading VQA architectures as of early 2019.

Agrawal *et al.* [77] introduce a benchmark dataset to test artificial intelligence algorithms with a broad set of images and human-engineered questions that are designed to challenge the intellect of a hypothetical toddler, alien, or smart robot. That is, the questions were designed to be non-obvious, requiring background knowledge and reasoning skills that current state-of-the-art systems may not possess. COCO (Lin *et al.* [78]), Visual7W (Zhu *et al.* [79]), and DAQUAR (Malinowski *et al.* [80]) are also commonly used in this field, and ask generic questions, usually from a human's perspective of the world. They contain images of human environments and the objects commonly found in them.

This work is more concerned with establishing the capability of a VQA-derived system to understand concepts related to navigation. This means identifying landmarks and their orientations in relation to a hypothetical robot. These concepts are not generally tested by the questions and environments found in the datasets mentioned above. We instead turn to the CLEVR dataset (Johnson *et al.* [25]), which is designed to test spatial and abstract reasoning for VQA systems.

While the task posed by each dataset is the same conceptually, the datasets are rather different. CLEVR uses realistic images that are nevertheless synthetic, with a limited number of objects in a single context. The questions are algorithmically generated from a set of templates. While the number of question variants is quite large, there are fundamentally just six kinds of questions. In robot planning, it is not unusual to have a limited, structured language reminiscent of the algorithmically generated CLEVR questions. In contrast, the Visual Genome dataset uses real images annotated by crowdsourcing. The questions and answers are more diverse than those from CLEVR and naturally will lead to different strategies for designing the network.

The CLEVR dataset has effectively been solved. For example, Yi *et al.* [74] perform better than 99 percent correct for all categories of questions. Further, this paper lists six other models that also perform better than 90 percent for all categories. Because CLEVR has been solved, it is reasonable to believe that many of the spatial, relational, and grounding skills required to answer CLEVR questions have been accurately modeled by many of the proposed architectures. However, it can also be said that these models probably over-fit the synthetic data and may not work as well with real images. While this is likely true, the objective of this project is to demonstrate that it is *possible* to fuse command parsing, image processing, and path generation into a single deep learning framework. Since the CLEVR architectures are more mature, there is more certainty in the intermediate outputs required to perform our task.

Table 3.2: Reference VQA implementations considered at the commencement of this work. Not all networks report CLEVR performance.

Name	**Year**	**Accuracy %**	**Reasoning Model**	**Language Model**	**Attention Model**	**Source Code**
Inferring & Executing Programs (IEP)[17]	2017	88.6-96.9	Dynamic, Modular	2-layer LSTM	Procedural	Torch
End-to-End Module Networks (N2NMN)[22]	2017	72.1	Dynamic, Modular	2-layer LSTM	Procedural	TensorFlow
Stacked Attention Networks (SAN)[3]	2016	68.5	Iterative, Monolithic	LSTM/CNN	Multiplicative co-attention	Theano
Multimodal Compact Bilinear Pooling (MCB)[81]	2016	51.4	Static, Monolithic	2-layer LSTM	Bilinear	CAFFE
Visual7W[79]	2016	-	Static, Monolithic	LSTM	Additive co-attention	Torch
Feature-wise Linear Modulation (FiLM) [24]	2018	97.7	Static, Monolithic	GRU	-	Torch

For this reason, we have carefully selected architectures that perform well with CLEVR but are also general and not engineered specifically to the dataset. We categorize the different algorithms by their architectures in Table 3.2. A monolithic architecture has no modules used to solve particular sub-problems, which is in contrast to a modular network that uses specialized sub-networks. A dynamic network will choose the appropriate module for each iteration of the problem-solving process. In contrast, an iterative network maintains the same architecture but applies it repeatedly, as in a traditional recurrent network. Co-attention is an iterative attention process whereby an attention map is updated via repeated operations on both the image and query data. Finally, a static network will generate an answer in a single time step, as in Figure 3.5.

Most of the architectures for CLEVR follow a similar structure based on the original paper in Johnson *et al.* [25]. They have a base network, usually a pre-trained variant of ResNet-101 (He *et al.* [82]) from which $14 \times 14 \times 1024$ dimensional mid-level features are extracted, an adapter (stem) that conditions the features for the VQA task in general, a network core for modulating the feature based on the input phrase, and a decoder that generates an answer.

The CLEVR dataset is algorithmically generated, and ground truth algorithms are available. Most architectures predict the correct sequence of operations to predict the algorithm from the query and then dynamically construct a network to solve that program. The prediction task is then two-fold; to predict the correct algorithm and then the correct answer.

This is not possible for the general questions and answers found in Visual Genome. For this reason, the architectures for Visual Genome in Table 3.1 all have static architectures without specialized sub-units. For most queries, there is no clear, logical set of operations to generate a robust program. Likewise, because we wish to learn paths from demonstration, given a command and an image, we, therefore, do not consider the models that attempt to reconstruct an underlying algorithm. This excludes DDRProg, Neural Symbolic VQA, and TBDNets.

In contrast to these networks, the bodies of FiLM and Compositional Attention Networks (MAC) are a static sequence of feature modulation blocks that alter the feature map with respect to the encoded query. This implies that it is not necessary to model the underlying logical process to predict the correct answers to the CLEVR dataset, which makes these architectures ideal candidates for our work. Further, they have no task-specific modules, such as color or shape recognition sub-nets, indicating the generality of these architectures.

Finally, both networks train very quickly and consistently in contrast to architectures that model the underlying algorithm used to generate the data. While it may seem that having the ground truth algorithms makes the task easier to learn, the recurrent network becomes non-differentiable at the point where the command encoding network selects the various modules to construct each network instance. To work around this problem, some have used techniques like REINFORCE (as in IEP Johnson *et al.* [17] or Neural Symbolic VQA Yi *et al.* [74]) to perform policy gradient descent.

FiLM satisfies our constraints of a fully differentiable network without specialized modules that are specific to a given task. As such, we feel it is a good choice as the core of our network. We will now delve into two networks that demonstrate simplified principles in VQA architectures and then introduce FiLM.

3.3.1 Basic VQA Architectures

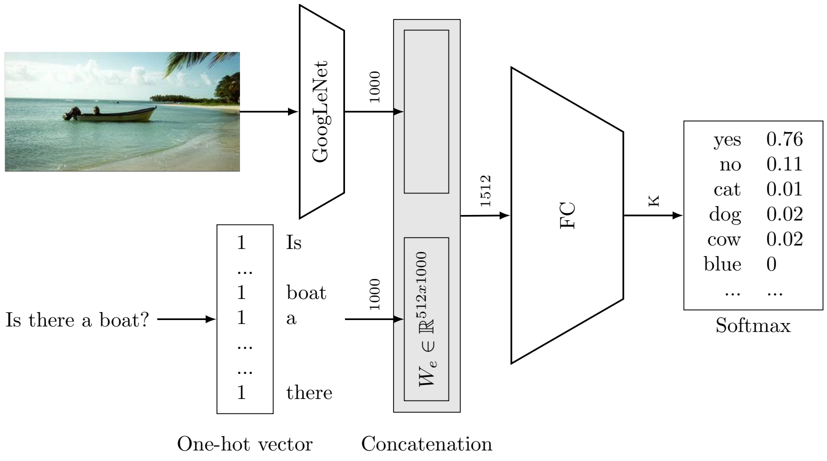


Figure 3.5: Basic VQA architecture adapted from Zhou *et al.* [2]. Image from SBD (Gould *et al.* [54])

Figure 3.5 is an example of a typical early system for solving this problem using a bag-of-words question representation (Zhou *et al.* [2]). Image features are supplied by high-performing classifiers that were pre-trained on other tasks, such as ResNet (He *et al.* [82]), GoogLeNet (Szegedy *et al.* [83]) or VGG (Simonyan *et al.* [84]). Features are extracted from the last layer of GoogLeNet in this purposely naïve baseline architecture. The feature vector is concatenated with the embedded query vector, and a shallow, fully connected network generates an answer to the query with a softmax output. The network is trained with cross-entropy loss.

However, this example network suffers from many limitations, two of which are critical to our task. First, the bag-of-words representation makes it difficult to encode complex queries. This is fatal for a navigation language, as even simple instructions, such as “Go to the left of the car that is next to the building” would have the same encoding as “Go to the left of the building that is next to the car.” Second, it cannot perform any kind of reasoning or attention process found in current high-performing architectures because there is no spatial information in the image features and no attention mechanism. This naïve implementation was unsurprisingly designed as a baseline and not as a competitive architecture.

A solution to the text encoding problem is to use the Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), or Gated Recurrent Unit (GRU) to produce a sequence-dependent representation. Figure 3.6 is an example of an architecture from Agrawal *et al.* [77] that uses LSTMs for language encoding. The final hidden state of the LSTM encodes the embedded query. For clarity, we have omitted that each word is first encoded as a one-hot vector that is embedded with a matrix W_e, followed by a tanh non-linearity, where $W_e \in \mathbb{R}^{300 \times N}$ and N is the number of words in the input vocabulary.

The word embedding matrix, W_e, can be learned, but in other models, the embedding is pre-

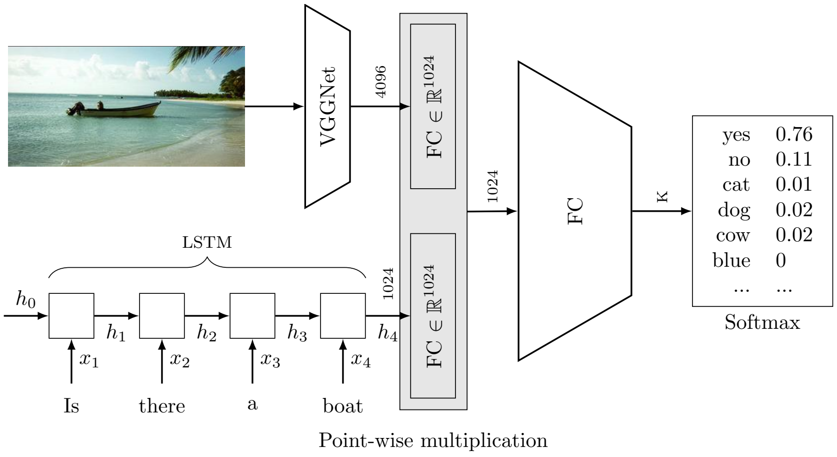


Figure 3.6: VQA architecture with LSTM text encoding adapted from Agrawal *et al.* [77]. Word embedding left out for clarity.

trained using GloVe (Pennington *et al.* [85]), word2vec (Mikolov *et al.* [86]), and others. The choice of encoding can affect results, as word2vec, for example, is designed to place semantically similar words, like "red" and "blue" in close proximity in the vector space, which may make discrimination between the two colors more difficult since they have a similar embedding.

Graves [87] demonstrates that LSTMs with larger hidden state perform better in machine translation tasks. This is reflected in VQA architectures such as Johnson *et al.* [17] and Hu *et al.* [22]. Agrawal *et al.* [77] try a two-layer LSTM and observe some benefit, especially when combined with L_2 normalization of the image features.

Alternatively, the hidden state may be used as it evolves over time. Xiong *et al.* [88] use the hidden state sequence from a bi-directional Gated Recurrent Unit (GRU) to assemble facts from both image and text features. Finally, Yang *et al.* [3] use a CNN for question encoding. However, it is not recurrent, and this network is limited to the three successive words of the CNN's receptive field, limiting its ability to encode complex structure in the query.

3.3.2 Attention Models

Spatial information is still missing from this architecture. A query such as "Is there a dog in the image?", may work, but a more complex question, such as, "Is there a dog next to the chair?" will not. The image feature vector is a flattened gestalt representation of the image. One solution is to use mid-level feature maps that retain spatial information. Attention mechanisms are another approach to solve this. The method presented here is a form of soft attention, where regions of the feature map are weighted based on their perceived importance.

Symbol grounding is essential to navigation, and therefore attention mechanisms are of particular interest to this work. However, FiLM in Section 3.3.3 exhibits good performance without an explicit attention model.

Visual7W

Visual7W (Zhu *et al.* [79]) is an example of a network that integrates spatial information using an LSTM through a multiplicative attention model for grounded VQA (Figure 3.7). They use a modified LSTM to encode a question and an image at the same time and then use a decoding step to generate an answer. Equations (3.14) to (3.17) include an additional term to the forget, input, and output, LSTM functions.

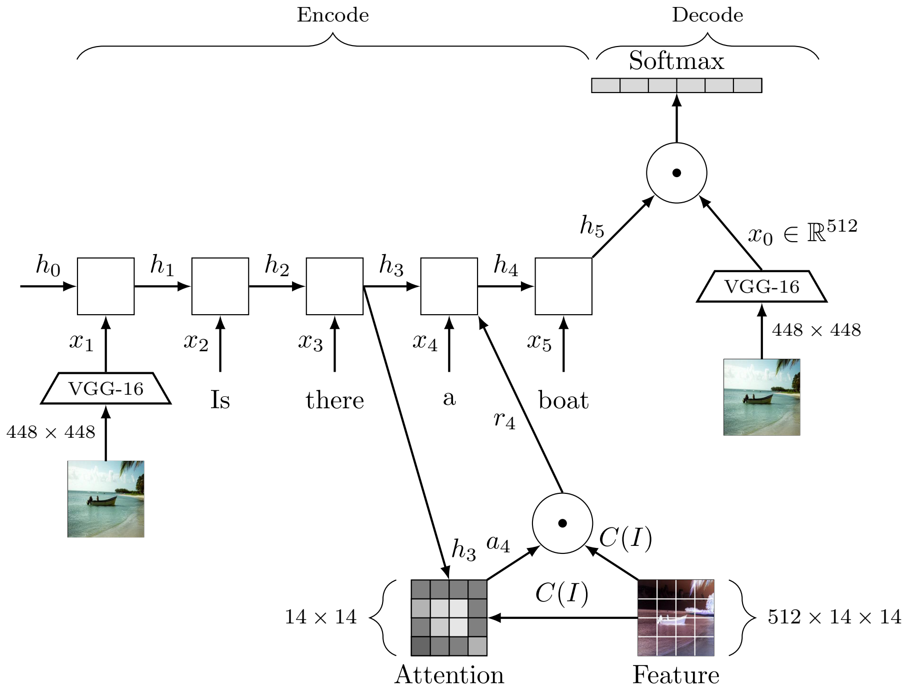


Figure 3.7: Simplified diagram of the Visual7W VQA network adapted from Zhu *et al.* [79]. The attention mechanism is integrated with the LSTM query encoding process. Not all attention map updates are illustrated for clarity.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + \underline{W_{rf} r_t} + b_f) \tag{3.14}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + \underline{W_{ri} r_t} + b_i) \tag{3.15}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + \underline{W_{ro} r_t} + b_o) \tag{3.16}$$

$$g_t = \phi(W_c x_t + U_c h_{t-1} + \underline{W_{rc} r_t} + b_c) \tag{3.17}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t \tag{3.18}$$

$$h_t = o_t \circ \phi(c_t) \tag{3.19}$$

$$h_0 = c_0 = 0 \tag{3.20}$$

This additional input, r_t, is generated by the attention mechanism, which is a function of the hidden state and the feature map $C(I) \in \mathbb{R}^{512\times 14\times 14}$ taken from the fourth convolutional layer of VGG-16. Equations (3.21) to (3.23) shows that the attention term is the weighted sum of the vectors in the feature map. Given the hidden state of the LSTM, h_{t-1}, which is dependent on the prior words in the questions and the image, the attention mechanism adjusts the weights of the image features to amplify those which are pertinent to the question.

$$e_t = w_a^T \phi(W_{hc}h_{t-1} + W_{cc}C(I)) + b_a \tag{3.21}$$

$$a_t = \text{softmax}(e_t) \tag{3.22}$$

$$r_t = a_t^T C(I) \tag{3.23}$$

Finally, to complete the architecture, the initial inputs and final output to the process are defined as:

$$x_0 = W_i F(I) + b_i \tag{3.24}$$

$$x_1 = W_w OH(t_1) \tag{3.25}$$

$$Z = \text{softmax}(F(I) \cdot h_n) \tag{3.26}$$

Here, $F(I)$ is the output of the last fully connected layer of VGG-16, and OH is the one-hot representation of the word t_1. $W_w \in \mathbb{R}^{512\times N}$ and $W_i \in \mathbb{R}^{512\times 4096}$ are embedding matrices, where N is the number of words in the vocabulary.

Stacked Attention Network (SAN)

Instead of integrating attention with question encoding, one can also make it an independent process, as in that Stacked Attention Network (SAN) from Yang *et al.* [3] (Figure 3.8). The additive attention mechanism in this model receives both the 512 × 14 × 14 feature map from the last pooling layer of VGG-16 and the encoded question, which can either come from an LSTM or a CNN. For brevity, we will not describe the CNN question encoding. While competitive, it is not common in the VQA literature.

$$h_A = \phi(W_{I,A}v_i \oplus (W_{Q,A}v_Q + b_A)) \tag{3.27}$$

$$p_I = \text{softmax}(W_P h_A + b_P) \tag{3.28}$$

Here, $v_I \in \mathbb{R}^{d\times m}$ where m are the number of image regions, in this case, 196 from a 14×14 feature map, and d is the dimension of the feature vector, 512. The feature map from VGG-16 is converted from a 3D tensor to a 2D array of feature vectors. In Equation (3.27), the input image representation v_I is embedded and combined with the question representation using the $\oplus$ operator.

The attention map is the generated in Equation (3.28), similar to Equation (3.22). Each image feature is weighted by this attention map and summed to compute a new, overall image representation

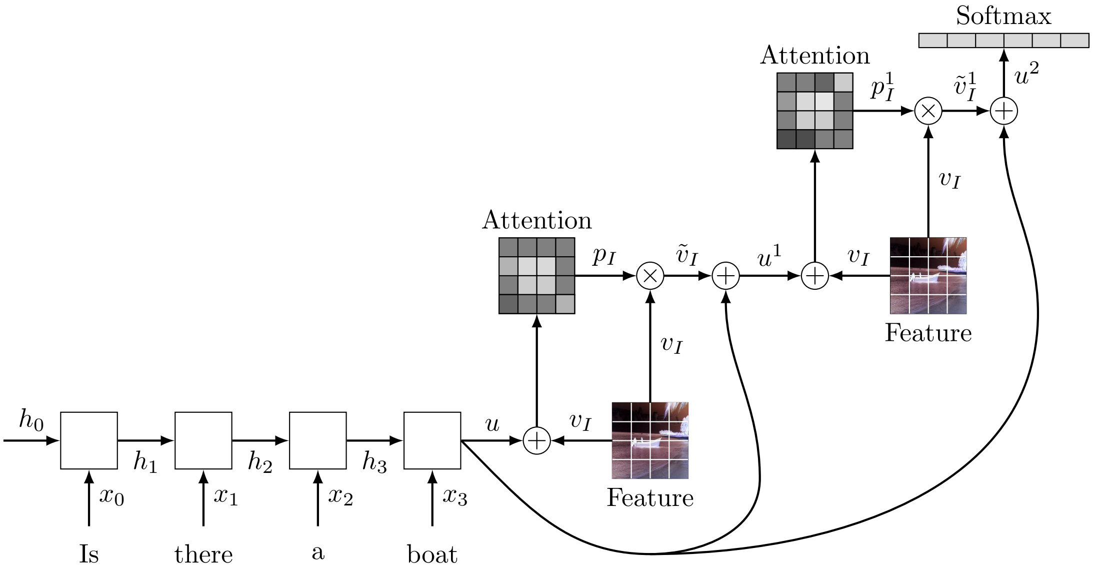


Figure 3.8: Simplified diagram of a Stacked Attention Network adapted from Yang *et al.* [3]. The attention mechanism is a separate process after input encoding. In this case, there are two attention layers.

vector (Equations (3.29) and (3.30)).

$$\tilde{v}_I = \sum_i p_i v_i \tag{3.29}$$

$$u = \tilde{v}_I + v_Q \tag{3.30}$$

This new representation, u, can be reprocessed to address queries that require iterative refinement, such as, “What is on the desk next to the chair next to the sofa?” (Equations (3.31) and (3.32)) The new representation, u^k is computed in a similar fashion in Equations (3.29) and (3.30).

$$h_A^k = \phi(W_{I,A}^k v_i \oplus (W_{Q,A}^k u_{k-1} + b_A^k)) \tag{3.31}$$

$$p_I^k = \text{softmax}(W_P^k h_A^k + b_P^k) \tag{3.32}$$

The final answer is computed as:

$$p_{ans} = \text{softmax}(W_u u^K + b_u) \tag{3.33}$$

We have seen attention via multiplication and addition, but there are other attention mechanisms such MutliModal Compact Bilinear Pooling (Fukui *et al.* [81]) that use an outer product.

3.3.3 Adapting FiLM for Cost Map Generation

FiLM (Perez *et al.* [24]) stands for Feature-wise Linear Modulation. This architecture performs well on the CLEVR dataset without an explicit reasoning model. This generic design makes FiLM adaptable to different domains. It uses a similar LSTM architecture for language as SAN and Visual7w but instead uses a simple and novel feature map modulation technique with no attention model. For this reason, we have chosen FiLM as the basis of our architecture.

The core element of FiLM is the FiLM-ed residual block (ResBlock) (Equation (3.36), Figure 3.9), which conditions the image features with an affine transform produced from the language input. These blocks are derived from the residual learning building blocks from He *et al.* [82]. Equation (3.36) describes how the FiLM layer of the ResBlock modulates the feature map by scaling and offsetting it with function $f_c(x_i)$ and $h_c(x_i)$, where i indexes the FiLM layer and c indexes the feature channel.

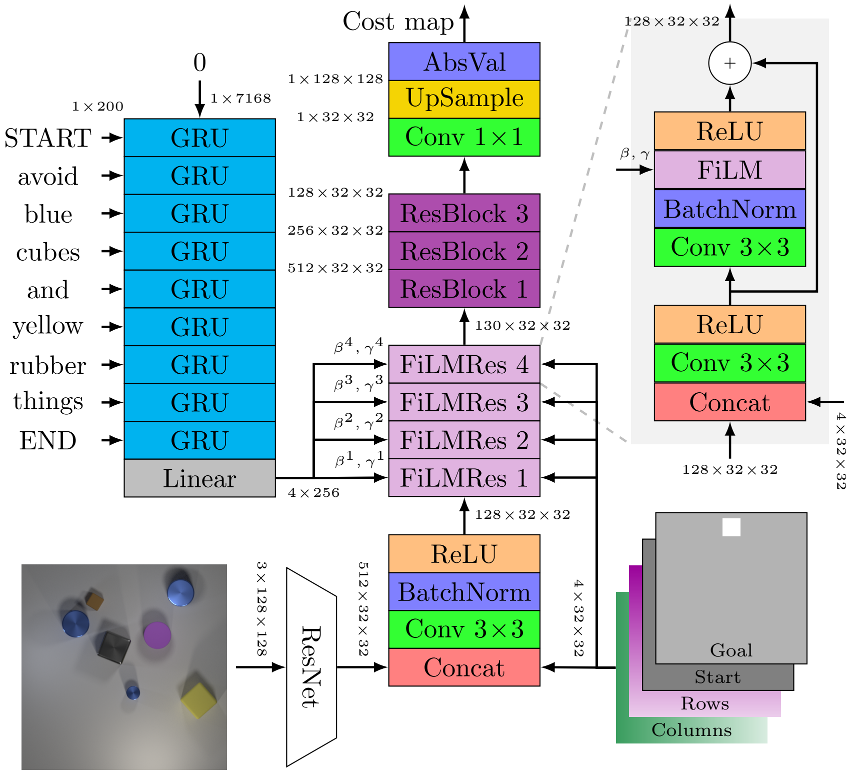


Figure 3.9: Deep Max-Margin Planning network architecture developed in this thesis.

$$\gamma_{i,c} = f_c(x_i) \tag{3.34}$$

$$\beta_{i,c} = h_c(x_i) \tag{3.35}$$

$$FiLM(F_{i,c}|\gamma_{i,c}, \beta_{i,c}) = \gamma_{i,c} F_{i,c} + \beta_{i,c} \tag{3.36}$$

FiLM uses pre-trained ResNet-101 mid-level features with a resolution of 14 × 14 × 128. They are

Command	How many green objects are there
Tokens	[<START>, how, many, green, objects, are, there, <END>]
Encoded	`[ 1 46 31 68 12 64 33 2 0 0 0 0 0 0 0 0 0 0 0 0 ]`

Table 3.3: Command transformed into tokens.

augmented with two channels that encode the row and column for each position of the feature map. The FiLM parameters $\gamma_{i,c}$ and $\beta_{i,c}$ do not include spatial information. Because FiLM only multiplies and offsets the feature map, it is computationally efficient and scales well with feature map size.

The functions $f_c(x_i)$ and $h_c(x_i)$ are the final hidden state of the input sequence encoder. FiLM uses a Gated Recurrent Unit (GRU), but may also use an LSTM, followed by a linear net to decode the hidden state into the FiLM vector. The GRU hidden state is $\mathbb{R}^{4096}$. The input query is tokenized (Table 3.3) into a fixed length string and embedded into a $\mathbb{R}^{200}$ vector using word2Vec (Mikolov *et al.* [86]). We have also successfully trained models using GloVe (Pennington *et al.* [85]). The hidden state after the last non-null token is taken as the output which is decoded into FiLM parameters. Since each of the 4 FiLMed ResBlocks scales and offsets a 128-channel feature map, the output of the language encoder must be $\mathbb{R}^{1024}$.

After feature modulation, FiLM uses three standard residual blocks, which reduce the number of channels in the feature map, followed by an output classifier. In our adaptation for navigation, we replace the classifier with a 1×1 convolution to reduce the number of channels, followed by an up-sampling to increase spatial resolution. An absolute value output function produces non-negative cost map values, required by planners such as Dijkstra's or A*. We augment the feature map with a one-hot encoded start and end location of the agent to assist with commands that implicitly reference the agent's starting location.

3.4 Metrics

Robot trajectories are the ultimate product of the system, making trajectory metrics an essential part of the performance evaluation. The cost function of the notional robot operator that we are trying to replicate is unknown and, in general, difficult to reconstruct by means other than by example. Therefore, the metrics used here are constrained to those which assume nothing about path cost or the latent cost function.

Two metrics are standard in robotics and computer vision literature: the Hausdorff metric and the Fréchet metric. The planning system generates polygonal curves, limiting our scope to suitably specialized versions of the metrics for a two-dimensional Euclidean space with piece-wise linear paths.

3.4.1 Hausdorff Distance Metric

For point sets P and Q in a Euclidean space, the Hausdorff metric can be written as:

$$d(p,q) = \|p - q\| \tag{3.37}$$

$$d_1(P,Q) = \max_{p \in P} \left(\min_{q \in Q} d(p,q) \right) \tag{3.38}$$

$$d_H(P,Q) = \max \left(d_1(P,Q), d_1(Q,P) \right) \tag{3.39}$$

Figure 3.10a is a graphical example of a one-sided pointwise Hausdorff metric. We will refer to the one-sided match of each element in P to the nearest neighbor in Q as the Hausdorff coupling.

Dubuisson *et al.* [89] examined variants of this metric for greater robustness in the presence of noise. In particular, the median of the minimum distance from points in P onto Q is defined as Equation (3.40). The definition of the modified Hausdorff distance metric is then stated in Equation (3.41).

$$d_{50} = {}^{50}K^{th}_{p \in P} \left(\min_{q \in Q} d(p,q) \right) \tag{3.40}$$

$$d_{MHD}(P,Q) = \max \left(d_{50}(P,Q), d_{50}(Q,P) \right) \tag{3.41}$$

We report the 90$^{\text{th}}$ percentile Hausdorff distance over a sample when computing aggregate metrics, denoted as ${}^{90}H_{GT}$ and ${}^{90}H_{SP}$ when compared to the ground truth path or the shortest path that avoids obstacles, respectively.

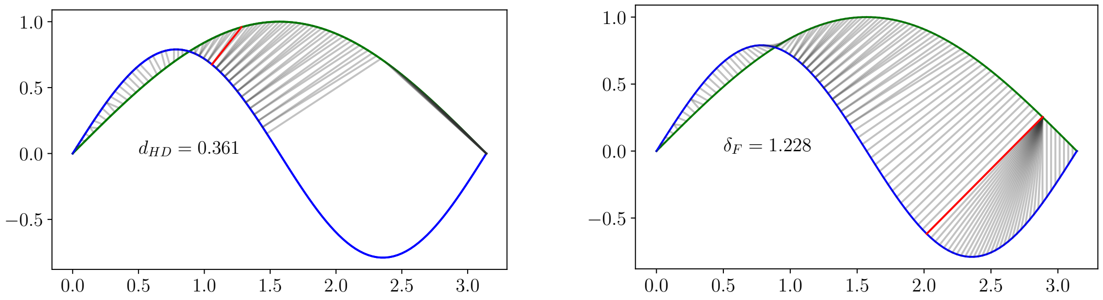
Dubuisson *et al.* [89] examined the utility of the Hausdorff distance metric (and others) as a metric for conformity between point sets in the presence of noise. They determined that the modified Hausdorff metric (MHD), Equation (3.41), is the most robust variant of the metrics studied. Huttenlocher *et al.* [90] use the Hausdorff metric as a way to align exemplar models to binary edge images. Shapiro *et al.* [91] also consider more general image transformations using an averaged one-sided Hausdorff distance (Equation (3.38)). It has since been used in many recent works relevant works for robot navigation and human path prediction, such as Wulfmeier *et al.* [66] and Lee *et al.* [63].

This algorithm is $\mathcal{O}(n)$ for discrete point sets, a great advantage for numeric optimization. However, while a low Hausdorff metric is a necessary condition to ensure point set conformity, it is hardly sufficient, as we shall discuss in Section 3.4.3.

3.4.2 Discrete Fréchet Distance Metric

The Fréchet Distance between continuous curves is defined as:

$$F(A,B) = \inf_{\alpha,\beta} \max_{t \in [0,1]} d(A(\alpha(t)), B(\beta(t))) \tag{3.42}$$



(a) One-sided pointwise Hausdorff computation from blue curve onto green curve. Couplings are in gray. Median coupling in red.

(b) Pointwise Fréchet coupling, gray. Maximal length coupling, the Fréchet distance, in red.

Figure 3.10: Example of Hausdorff and Fréchet couplings

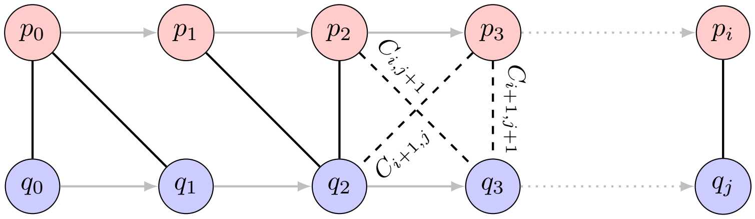


Figure 3.11: Example Sequence Walk problem with potential couplings (dashed) corresponding to Equation (3.46).

where A and B are continuous curves parameterized by functions α and β on the interval of $[0, 1]$. Also called the dog leash distance, the intuitive explanation of this metric is that each point in one curve must be matched to a point in the other. We move along the curves monotonically using the parameterized function $\alpha(t)$ and $\beta(t)$, one being the dog and the owner the other. Either may stop or slow down, but they can never go back or skip ahead on the curve. Among all the possible traversals of the curves, determined by some $(\alpha(t), \beta(t))$, the Fréchet distance is the one with a maximum leash length less than all others (Figure 3.10b). This metric is difficult to compute analytically for arbitrary continuous curves.

The discrete Fréchet distance metric is a variant that measures conformity between point sequences. Introduced by Eiter *et al.* [92], it has a compact dynamic programming solution (Equation (3.47)) with a naïve runtime of $\mathcal{O}(n^2)$[1], and is closely related to other sequence walk problems, such as Dynamic Time Warping and Longest Common Sub-sequence. At each timestep, the algorithm chooses to advance the position of the owner, the dog, or both (Section 3.4.2). This metric was used by Oh *et al.* [11] for evaluating paths used for robot navigation.

$$C_{0,0} = d(p_0, q_o) \qquad \text{Base Case} \tag{3.43}$$

$$C_{i,0} = \max\left(d(p_i, q_0), C_{i-1,0}\right) \qquad \text{Edge Case 1} \tag{3.44}$$

[1] We approximate $n \approx |P| \approx |Q|$.

$$C_{0,j} = \max\left(d(p_0, q_j), C_{0,j-1}\right) \qquad \text{Edge Case 2} \tag{3.45}$$

$$C_{i,j} = \max\left(d(p_i . q_j), \min(C_{i,j-1}, C_{i-1,j}, C_{i-1,j-1})\right) \qquad \text{General Case} \tag{3.46}$$

$$\delta_F = C_{|P|-1,|Q|-1} \qquad \text{Discrete Fréchet Metric} \tag{3.47}$$

The Fréchet distance between polygonal curves is more complicated than a sequence walk upon the vertices of P and Q. This is because the shortest distance between a point $p_i \in P$ may fall in the interval $[q_j, q_{j+1}] \in Q$. Eiter *et al.* [92] show that the error between the discrete Fréchet distance metric for point sequences (δ_{dF}) compared to the true Fréchet distance for the associated polygonal curve (δ_F) is bounded by the maximal distance between any two sequential points in P or Q (Equation (3.48)).

$$\delta_{dF}(P, Q) \leq \delta_F(P, Q) + \max\left(D(P), D(Q)\right) \tag{3.48}$$

$$D(P) = \max_{i=2\ldots|P|} d(p_i, p_{i-1}) \tag{3.49}$$

$$D(Q) = \max_{i=2\ldots|Q|} d(q_i, q_{i-1}) \tag{3.50}$$

In practice, we interpolate polygonal curves such that $\delta_F(P, Q) \leq= 0.1$pixels. This means that the size of the dynamic programming table increases by a factor of 100. For a $10^3 \times 10^3$ pixel image, we can expect a path to be approximately 10^3 pixels long. The approximate table size is then $\left(10^3 \times 10\right) \times \left(10^3 \times 10\right) = 10^8$ entries. While the table is large, each entry is visited just once, with a systematic expansion that leads to good memory locality and is well within the limits of modern desktop computers. If the Fréchet couplings themselves are not required, only the frontier nodes need to be stored in memory. The entire computation will easily fit within the cache of most CPUs or the register file of a GPU streaming multiprocessor.

After termination of the recurrence in Equation (3.46), the matches between points in P and Q, or couplings, are determined by back-tracking from $C_{N-1,M-1}$, following the path of $\min(C_{i,j-1}, C_{i-1,j}, C_{i-1,j-1})$ and recording the coordinates (i, j) at each step. We report the 90^{th} percentile coupling distance over a sample when computing aggregate metrics, denoted as ${}^{90}F_{GT}$ and ${}^{90}F_{SP}$ when compared to the ground truth path or the shortest path that avoids obstacles, respectively.

Alt *et al.* [93], Agarwal *et al.* [94], and others have improved runtime slightly. For example, Agarwal *et al.* [94] reduces runtime to $\mathcal{O}(\frac{n^2 \log\log n}{\log n})$ by using geometric principles to eliminate non-viable solutions, but this increases the complexity of a practical implementation. This is particularly problematic for GPU architectures that are unable to efficiently process divergent code, and therefore, the basic dynamic programming definition is the most viable choice for this work.

3.4.3 Discussion

The Hausdorff metric is simple to compute ($\mathcal{O}(n)$) in comparison to the Fréchet metric ($\mathcal{O}(n^2)$). The Hausdorff metric also has a simple form, which makes it attractive for integration into more complex

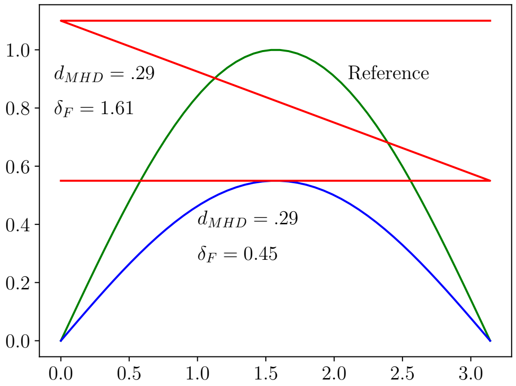


Figure 3.12: Modified Hausdorff Distance of red curve to green curve is the same as the MHD from blue curve to green curve.

algorithms such as Max Margin Planning (Section 3.2.1). However, the simplicity of the Hausdorff metric comes at the cost of it being less discriminant. The red a blue curves in Figure 3.12 have the same modified Hausdorff distance, despite the red curve's radically different shape. Figure 3.10a lends some insight as to how this is possible. The shape of the blue curve when $y < 0$ is completely irrelevant to the metric and can be made arbitrarily negative without affecting the metric. In contrast, in Figure 3.10b every point is visited and contributes to the metric.

3.5 Summary

This chapter introduced the theory and related works that support our architecture. The MDP formulation of the planning problem is distinct from other probabilistic graph formulations used in robot navigation, such as Kollar *et al.* [14], primarily because of the Markovian assumption. This simplification gives us access to a great wealth of prior work in MDPs. We use a map representation with a finite time horizon. Since the reward function is undefined, we use Inverse Optimal Control to imitate an expert agent using samples of its behavior.

Ng *et al.* [61] note that Inverse Reinforcement Learning is ill-posed; there is generally not enough information in the data to reconstruct a unique solution. We have explored three techniques that add constraints sufficient to produce a usable cost map. Ratliff [58] uses the max-margin hypothesis, and Ziebart [59] uses the principle of maximum entropy. We choose MMP because it is less computationally intensive since it uses an efficient optimizer in the form of A* or Field D*. However, it is entirely possible to perform similar research with MaxEnt IRL, and with our transition to a GPU planner based on the Bellman-Ford algorithm in Section 5.3, the runtime difference between MaxEnt IRL and MMP is much more similar.

We review existing network architectures for VQA that were state-of-the-art at the commencement of this work and study how the methods they use to fuse language and vision to answer questions can adapt to generate cost maps. We choose FiLM because it is model-free. This work is relevant precisely because it is difficult to model human planning explicitly, just as it was difficult to model human object detection in the past. Deep learning has largely replaced model-based vision methods, and they will likely do the same for robot navigation. The next chapter will explore how Max Margin Planning acts as a loss function for a simple problem and develop the tools required to combine it with a deep network.

Chapter 4

Foundational Experiments

In this chapter, we introduce our solution for robot navigation, as stated in Section 1.1. We begin by applying the Maximum Margin Planning (MMP) to a reduced problem (Sections 4.1 and 4.2). Since it is not a loss function commonly used for deep learning, it is necessary to understand how it behaves in a simple context before moving to a complex model. Li *et al.* [95] use a related SVM margin loss to train a 3D human pose estimator. The loss equation specified by Ratliff *et al.* [20] must be carefully translated into a computational algorithm that is stable over millions of iterations. We have identified problems attributable to numerical stability and offer solutions to make the algorithm practical.

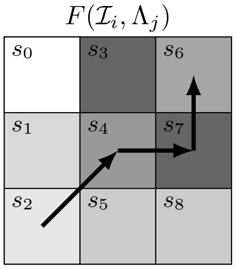
4.1 Adapting MMP for Deep Learning

Ratliff [58] finds the w which minimizes the loss in Equation (3.13) using a sub-gradient descent method. We adapt this method to our problem by first incorporating the weight vector w as part of the non-linear function $F(\mathcal{I}_i, Z_j; \theta)$[1]. This is a deep network, which accepts as input an image $\mathcal{I}_i$ with attribute Z_j, and returns a $d \times |\mathcal{S}||\mathcal{A}|$ matrix of d dimensional feature vectors for *each* state/action pair. This is not necessarily a convex problem, so the sub-gradient optimization is not guaranteed to converge. However, that is also true for the gradient descent methods employed in deep learning in general.

The states $\mathcal{S}$ are positions on a $n \times m$ map, and the actions are moves to the 8-connected neighbors of any location. Therefore, our μ are $8|n||m|$ dimensional sparse vectors, which would imply that $F(\cdot)$ is also $8|n||m|$ dimensional. However, our actual model for F is simpler than it would first appear.

In its non-vectorized form, F is a map of the cost of traversing a region of space. Actions that move horizontally or vertically incur one-half of the current cell plus one-half the cost of the new cell. When moving diagonally, this cost increases by a factor of $\sqrt{2}$ to account for the longer distance

[1] In future references to F, we may omit the parameters such as θ, Λ, etc., for brevity.



(a) Example trajectory and cost map.

$$\text{Action}$$

$$\text{State}\quad\begin{array}{c}\\ s_0\\ s_1\\ s_2\\ s_3\\ s_4\\ s_5\\ s_6\\ s_7\\ s_8\end{array}\begin{array}{c}\begin{array}{cccccccc}\uparrow & \leftarrow & \downarrow & \rightarrow & \nwarrow & \swarrow & \searrow & \nearrow\end{array}\\ \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{2}\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{\sqrt{2}}{2}\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}\end{array}$$

(b) State/action visitation.

$$\begin{bmatrix}0\\ 0\\ \frac{\sqrt{2}}{2}\\ 0\\ \frac{1}{2}+\frac{\sqrt{2}}{2}\\ 0\\ \frac{1}{2}\\ \frac{1}{2}+\frac{1}{2}\\ 0\end{bmatrix}$$

(c) State visitation, μ.

Figure 4.1: Transformation of path into state visitation count, μ, for path between 8-connected neighbors.

traveled. The state/action matrix (Figure 4.1b) can be quite large but is also sparse (Figure 4.1a). This matrix is readily condensed into a weighted vector of dimension $|\mathcal{S}|$, which we define as the state visitation, μ[2]. This process is described by Algorithm 1.

With feature map $F(\cdot)$ an $|n||m| \times 1$ vectorized cost map the path cost is succinctly expressed by the linearly separable Equation (4.1).

$$C(\mu) = F(\cdot)^T \mu \tag{4.1}$$

The objective is now Equation (4.2), with the sub-gradient for $q = 2$ in Equation (4.3) and the sub-gradient for $q = 1$ in Equation (4.4). As in Ratliff *et al.* [20], the minimizer for the loss-augmented cost map (Equation (4.5)) is found efficiently using Dijkstra's algorithm or A*. However, in our experience, Dijkstra's algorithm is more appropriate since the cost map may be very noisy, especially in the early phases of training, and any heuristic to guide the search will not improve performance or violate assumptions necessary for correctness. While efficient, optimizers such as Dijkstra's algorithm prohibit the agent from visiting the same state more than once, limiting the paths the algorithm can learn.

$$L = \min_{\theta}\left(\frac{1}{N}\sum_i \frac{\beta_{ij}}{q}\left\| F(\mathcal{I}_i, Z_j;\theta)\mu_{ij} - \min_{\mu\in\mathcal{G}_i}\left(\underbrace{F(\mathcal{I}_i, Z_j;\theta) + \ell(\mu_{ij})^T}_{\text{loss-augmented cost map}}\right)\mu\right\|^q + \frac{\lambda}{2}\|\theta\|^2\right) \tag{4.2}$$

[2] Ratliff [58] specifies a more general form of F, which specifies different costs for each state and action. In our simplified problem, the cost for traversing a map cell is isotopic.

Algorithm 1 Path visitation for A* paths. P is a sequence of path coordinates. M_r is the number of rows in the map. $P_r(i)$ and $P_c(i)$ access the row and column of the i-th path element.

```
function PathVisitation($P$, $M_r$)
    V ←0                                                  ▷ Zero vector ∈ $\mathbb{R}^{M_r M_c}$
    v ←0
    for $i$ in 0...$|P| - 1$ do
        $r_0 \leftarrow P_r(i)$
        $c_0 \leftarrow P_c(i)$
        if $i \neq |P| - 1$ then
            $r_1 \leftarrow P_r(i+1)$
            $c_1 \leftarrow P_c(i+1)$
            if $r_0 \neq r_1$ and $c_0 \neq c_1$ then
                v ←$\frac{\sqrt{2}}{2}$                   ▷ Diagonal path segment
            else
                v ←$\frac{1}{2}$                          ▷ Vertical/Horizontal path segment
            end if
        end if
        $V(r_0 + M_r c_0) \leftarrow V(r_0 + M_r c_0) + v$  ▷ Accumulate visitation count
    end for
    return $V$
end function
```

$$\frac{\partial L}{\partial F} = \frac{1}{N}\sum_{i=1}^{N}\beta_{ij}\left(F(\mathcal{I}_i, Z_j; \theta)\mu_{ij} - (F(\mathcal{I}_i, Z_j; \theta) + \ell(\mu_{ij})^T)\mu^*\right) F(\mathcal{I}_i, Z_j; \theta)(\mu_{ij} - \mu^*_{ij}) + \lambda\theta \tag{4.3}$$

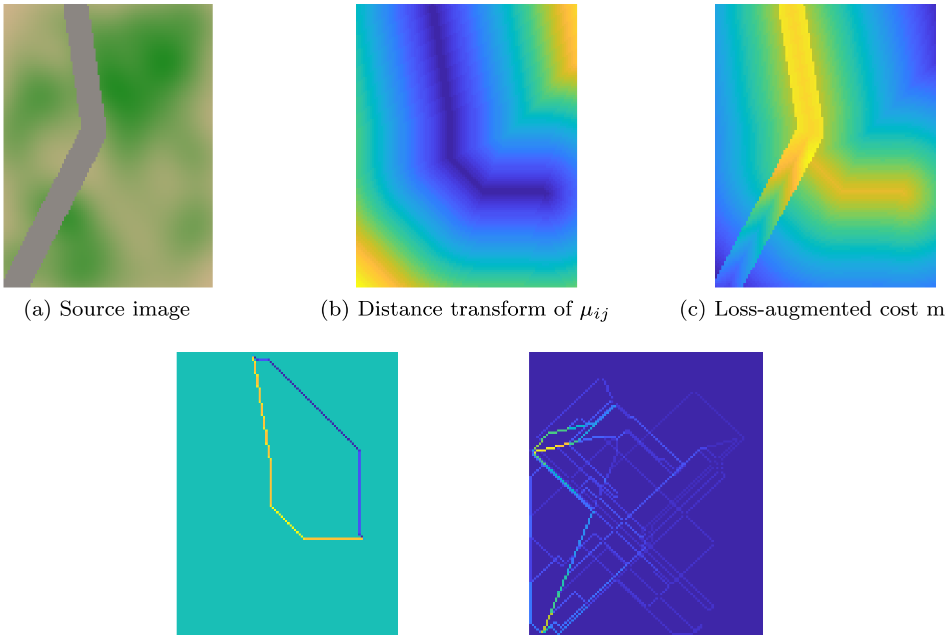
$$\frac{\partial L}{\partial F} = \frac{1}{N}\sum_{i=1}^{N}\beta_{ij}\,\operatorname{sgn} F(\mathcal{I}_i, Z_j; \theta)^T \circ (\mu_{ij} - \mu^*_{ij}) + \lambda\theta \tag{4.4}$$

$$\mu^* = \underset{\mu\in\mathcal{G}_i}{\operatorname{argmin}}(F(\mathcal{I}_i, Z_j; \theta) + \ell(\mu_{ij})^T)\mu \tag{4.5}$$

The gradients in Equations (4.3) and (4.4) are a $|\mathcal{S}| \times 1$ vector of derivatives for each state in the map and only containing corrections where μ and μ^* differ and is therefore sparse. The regularization term in Equation (4.4) is handled by the deep learning framework. We keep it here for consistency.

Figures 4.2a to 4.2d demonstrates the process of computing the loss for a single update for a single path. The source image is synthetic (Figure 4.2a). The distance transform is computed around the ground-truth path (Figure 4.2b), exaggerated for clarity which is then used to generate the loss-augmented cost map in Figure 4.2c. Using the same endpoints, a planner generates μ^* (not depicted). This is in turn used to produce the L_1 loss gradient in Figure 4.2d. The gradient increases the cost of cells where μ^* should not visit and decreases the cost of the cells that it should.

The loss gradient in Figure 4.2d is sparse. We combine a mini-batch of examples for each backward pass of the network to form a combined gradient (Figure 4.2e). This gradient is still sparse, with the majority of states not visited. See Algorithm 2 for a summary of the process.



(a) Source image

(b) Distance transform of μ_{ij}

(c) Loss-augmented cost map

(d) Loss gradient

(e) Accumulated loss gradient

Figure 4.2: Visualization of MMP planning loss.

4.2 Max Margin Planning with a Simple Network

Consider the simplified problem of reconstructing a cost map that is the linear combination of three cost maps encoding roads (low cost), dirt fields (medium cost) and vegetated areas (high cost), illustrated in Figure 4.3. Given a number of demonstration paths and the three cost map components in $F(\mathcal{I}) \in \mathbb{R}^{3\times|S|}$ we wish to recover $w_c \in \mathbb{R}^3$ such that the original cost map, $F' = w_c^T F(\mathcal{I})$.

We restate the MMP loss from Equation (4.6), with the addition of w_c to make the cost map a linear function of the features. There is no language input, $\Lambda_i$and the cost map is independent of the path starting and ending location.

$$L = \min_{w_c} \left(\frac{1}{N} \sum_i \frac{\beta_i}{q} \left\| w_c^T F(\mathcal{I}_i)\mu_i - \min_{\mu \in \mathcal{G}_i} \left(w_c^T F(\mathcal{I}_i)\mu + \ell_i(\mu_i)^T \mu \right) \right\|^q + \frac{\lambda}{2} \|w_c\|^2 \right) \tag{4.6}$$

$$g_{w_c} = \frac{1}{N} \sum_{i=1}^{N} \beta_i \left(w_c^T F(\mathcal{I}_i)\mu_i - (w_c^T F(\mathcal{I}_i) + \ell_i(\mu_i)^T)\mu^* \right) F(\mathcal{I}_i)(\mu_i - \mu^*) + \lambda \left\| w_c \right\| \tag{4.7}$$

$$g_{w_c} = \frac{1}{N} \sum_{i=1}^{N} \beta_i F(\mathcal{I}_i)(\mu_i - \mu^*) + \lambda \left\| w_c \right\| \tag{4.8}$$

$$\mu^* = \operatorname*{argmin}_{\mu \in \mathcal{G}_i} (w_c^T F(\mathcal{I}_i) + \ell_i(\mu_i)^T)\mu \tag{4.9}$$

Algorithm 2 Psuedo-code for Max Margin Planning with a deep network. w are network weights. $\mathcal{I}$ are map images. μ are expert paths of $|\mathcal{S}| \times |\mathcal{A}|$. Λ are language inputs. K is a spatial normalization constant, typically set to the map diagonal in pixels.

```
function DeepMMPLoss($w$,$\mathcal{I}$,$\mu$,$\Lambda$,$q$)
    $L \leftarrow 0$
    $dLdF \leftarrow 0$
    $F \leftarrow$Forward($w$,$\mathcal{I}$, $\mu$, $\Lambda$)                ▷ Network forward pass
    for $F_i$, $\mu_i$ in $F$, $\mu$ do
        $len \leftarrow$PathLength($\mu_i$)
        $\beta_i \leftarrow \frac{K}{len}$
        $A \leftarrow F_i + \ell(\mu_i)$                          ▷ Loss-augmented cost map
        $\hat{\mu}_i \leftarrow$Dijkstra($A$, $\mu_i$)          ▷ Cheapest path with same endpoints as expert
        $\mu_i^* \leftarrow$PathVisitation($\hat{\mu}_i$)     ▷ Convert to vectorized state visitation count
        $\mu_i \leftarrow$PathVisitation($\mu_i$)
        $L_x \leftarrow \beta_i \left\| \mathrm{vec}(F_i)^T \mu_i - \mathrm{vec}(A)^T \mu_i^* \right\|$
        if $q = 1$ then
            $L \leftarrow L + L_x$                              ▷ L$_1$ loss and gradient
            $dLdF \leftarrow dLdF + \beta_i \,\mathrm{sgn}(\mathrm{vec}(F_i))(\mu_i - \mu_i^*)$
        else
            $L \leftarrow L + \frac{1}{2}(L_x)^2$                      ▷ L$_2$ loss and gradient
            $dLdF \leftarrow dLdF + \beta_i L_x (\mu_i - \mu_i^*)$
        end if
    end for
return $\frac{L}{|\mathcal{I}|}$, $\frac{dLdF}{|\mathcal{I}|}$
end function
```

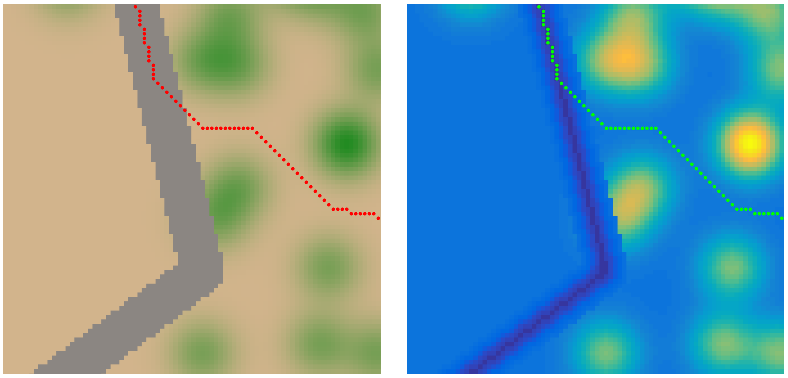
The update equation for w_c is Equation (4.10) for this trivial example.

$$w_c = w_c + \eta g_{w_c} \tag{4.10}$$

The cost maps are randomly generated with 2D Gaussians as hills, a randomly generated road that crosses the scene, and a fixed cost for all other regions (Figure 4.3a). All paths end at the edge of the scene on a point on the road and start at random locations in the interior. The correct solution to this diagnostic problem has zero error. Dijkstra's algorithm computes the ground truth paths with maps cells connected to their eight neighbors. There are 250 maps with a resolution of 128 × 128, each with about 64 example paths, with two-thirds of the maps used for training and one-third used for testing. Since this is a diagnostic dataset, there is no hold-out data.

We use the cost map components as the input to a simplified loss for initial experiments. The correct solution to this diagnostic problem has zero error. Later, we use the synthetic RGB image as input to a deep network to generate features from image data.

For the following experiments, we use a learning rate of $\eta = 10^{-4}$, $\lambda = 10^{-2}$, and use the absolute value function to generate the cost map. We use a distance transform from the ground truth path to each map cell to produce the augmented cost map. We train with stochastic gradient descent (SGD) with mini-batches composed of 10 percent of the available paths for one cost map at each



(a) RGB image.

(b) Learned cost map.

Figure 4.3: Example from dataset used in experiments (128×128 resolution). Road (grey), dirt(brown), vegetation(green) have increasing travel cost. Red path (left), ground truth. Green path (right), inferred path.

iteration. The cost maps are randomly permuted at each epoch.

4.2.1 Compensated Dot Product

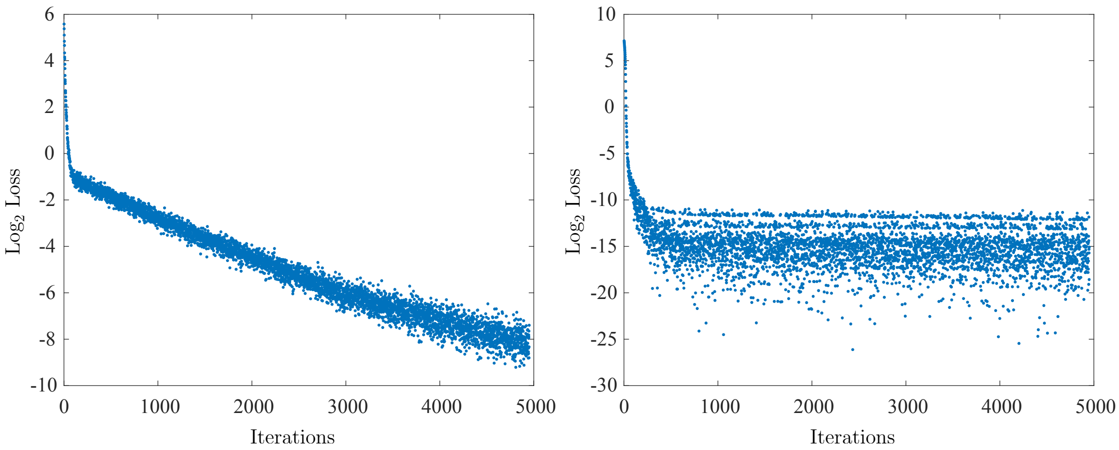
Given weights w_c drawn from the standard uniform distribution, we find that the algorithm often diverges with large gradients when using the L_2 version of the loss, $q = 2$ in Equation (4.6). Therefore, we begin by examining the L_1 loss ($q = 1$) and discover the loss itself is very noisy (Figure 4.4a).

Consider a test image of 1000×1000 pixels. It is reasonable to assume that a path in this image may be 1000 steps long. Stochastic gradient descent computes the loss gradients using random mini-batches of paths, perhaps about 100 examples. The loss in Equation (4.7) is a dot product between the weighted features and the path visitation count, $w_c^T F(\mathcal{I}) u_i$. This is a sum of 100000 floating-point numbers across all mini-batches.

Significant numerical errors may arise when adding many small values when performed without consideration for the practical aspects of floating-point addition. When accumulating a sum of many small numbers, the accumulator may have a significantly larger magnitude than an addend. The difference in magnitude between the two numbers limits the precision at which the smaller number can be expressed internal to the addition, potentially even rounding it to zero and resulting in a loss-of-precision error. This error is especially problematic for the L_2 case of Equation (4.7) because this accumulation is used to scale the gradient. This is not a problem for the L_1 version of the loss. The optimization is not stable without careful implementation, even if the equation is correct.

We use a compensated summation algorithm from Ogita *et al.* [96] to compute the terms in the loss equation[3]. Like the classic Kahan-Babǔshka ([98], [99]) or Knuth ([100]) algorithms for summation, Algorithm 3 compensates for numerical error by utilizing additional variables to store

[3] The Knuth TwoSum algorithm in this paper is incorrect. The last line should read $y = (a - (x - z) + (b - z)$. This was noted by Simon [97].



(a) Uncompensated dot product.

(b) Compensated dot product.

Figure 4.4: L$_1$ loss with uncompensated dot product (left) is more variable and has slower convergence compared to loss with compensated dot product (right). Linear pattern in compensated plot are individual outlier paths.

less significant mantissa bits that would normally be discarded. It is important to be certain that the compiler does not perform algebraic simplifications that overlook the subtleties of this algorithm.

Algorithm 3 Compensated dot product from Ogita *et al.* [96].

```
function CompensatedDotProduct($A$, $B$)
    $s \leftarrow 0$
    $y \leftarrow 0$
    for $a, b$ in $A, B$ do
        $a \leftarrow a * b$                                    ▷ Product
        $x \leftarrow s + a$                                    ▷ Sum
        $z \leftarrow x - a$
        $y \leftarrow y + ((a - (x - z) + (s - z))$             ▷ Residual
        $s \leftarrow x$
    end for
    $s \leftarrow s + y$                                        ▷ Compensated dot product
    return $s$
end function
```

Before resorting to this algorithm, we had also considered using 64-bit precision variables. Another solution is to sort the summands and either add them sequentially, from smallest to largest, or a pairwise reduction of summands with similar magnitude. We reject these approaches for two reasons. Sorting algorithms create additional workload with program branching and unpredictable memory access. The additional cost of a few more 32-bit operations is low for modern CPUs and GPUs with tremendous floating-point throughput. GPUs generally have poor 64-bit floating-point performance. The compensated dot-product allows single-precision operations to emulate double-precision arithmetic with modest additional runtime costs.

Figure 4.4b demonstrates how the compensated dot product reduces the noise when computing the L_1 loss. The version with the compensated dot product converges faster and can reduce loss by many orders of magnitude when compared to Figure 4.4a. The simplified problem presented here has an exact solution, although zero loss is rarely achievable in practice.

We are careful to note that the cost map and the MMP loss function do not have a fixed scale. Many cost maps with varying scales can have zero loss if they induce the planner to imitate the ground-truth path correctly. Consequently, incorrect predictions may have a loss with a varying scale from one run to the next but generally are well within the same order of magnitude, and the results are therefore comparable.

4.2.2 Huber Loss

The compensated summation algorithm helps to stabilize the L_2 loss. However, the L_2 loss is often unable to reliably converge given a random initial state of the network. Large errors at initialization produce even larger numbers when squared, leading to a numeric overflow. Further, the distance transform is not well suited as an error metric as it is the same as using a generalized Hausdorff coupling as a loss function (Figure 3.12). It does not accurately reflect the error for extreme cases found at the start of optimization.

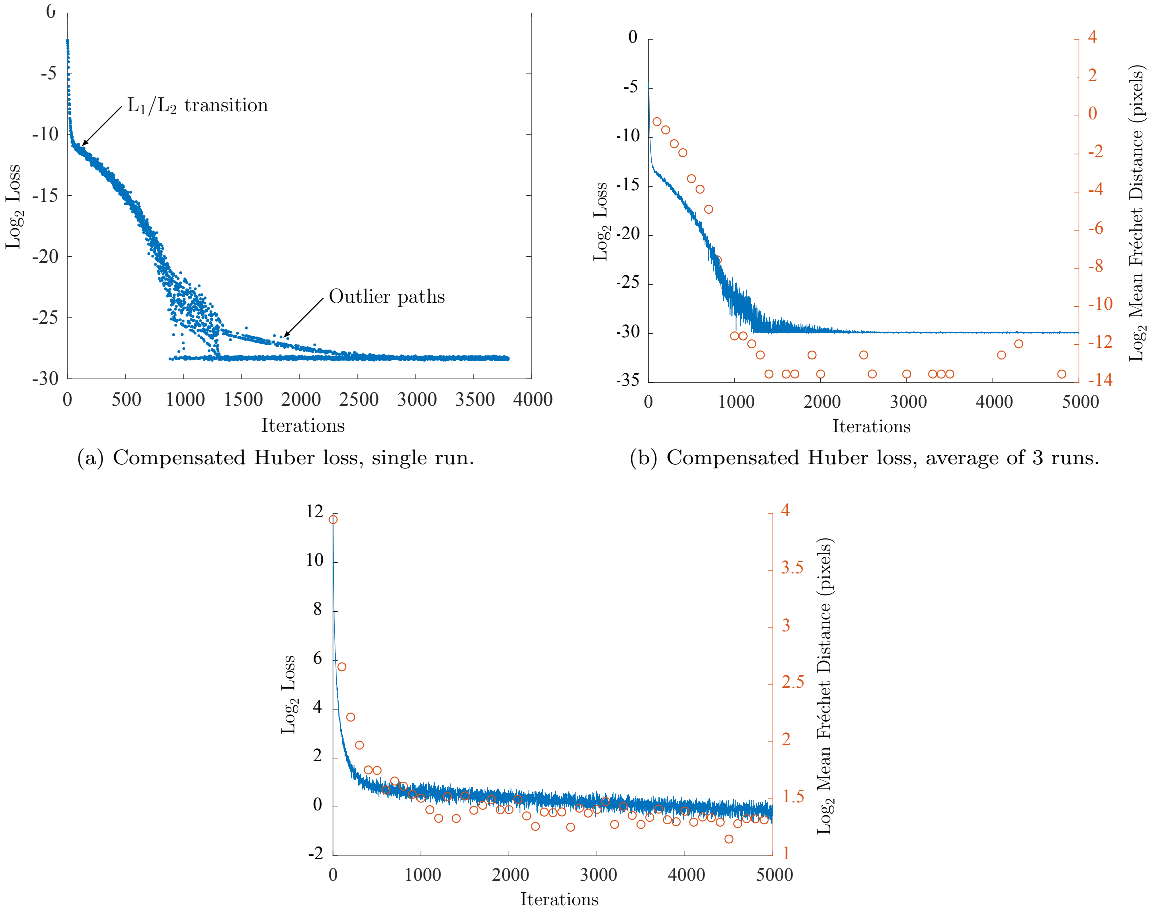
In contrast, the L_1 loss is less noisy and converges reliably but does not always reach a reasonable minimum. In the L_1 mode, the MMP loss increases the costs of cells that should not have been visited by a fixed amount and decreases the cost of cells that should have been visited. It does not use the loss function to scale the correction. Early in the optimization, this is an effective strategy when the loss function is a poor approximation of the Fréchet coupling metric. However, the distance transform is a good approximation of the Fréchet metric for small errors, which is why it is better to use the L_2 loss after the optimization has reached initial convergence.

We use the Huber loss ([101]) to combine the best features of the L_1 and L_2 versions of MMP (Equation (4.11)). The Huber loss transitions smoothly from L_1 loss to an L_2 loss when $L \leq \delta$.

$$L_\delta(L) = \begin{cases} \frac{1}{2}L^2 & |L| \leq \delta \\ \delta\left(|L| - \frac{1}{2}\delta\right) & |L| > \delta \end{cases} \tag{4.11}$$

In the following experiment, we use the same dataset and model from Section 4.2.1, applying the Huber loss to each training example individually, instead of on the mini-batch as a whole, so that the losses from individual examples have similar scale. The error gradient is similarly accumulated using the appropriate version for the Huber loss case selected for each training example. The regularizer in Equation (4.6) is not affected. We choose $\delta = 1e - 3$ and a learning rate of $1e - 1$.

Figure 4.5a shows a clear transition from an aggressive L_1 optimization to a more precise L_2 optimization. It also shows individual traces of outlier paths during the terminal phase of the minimization as the loss approaches zero. Figure 4.5b is an average over three runs and shows that



(a) Compensated Huber loss, single run.

(b) Compensated Huber loss, average of 3 runs.

(c) Compensated L$_2$ loss.

Figure 4.5: Huber loss allows aggressive corrections for the initial phase of optimization without divergence. The loss floor in Figures 4.5a and 4.5b is artificial, set to prevent underflow errors.

the Fréchet distance metric follows the Huber loss. Figure 4.5c is a trace averaging three L$_2$ runs with the largest learning rate that does not diverge. After 5000 iterations, the L$_2$ loss is far behind the Huber loss.

4.2.3 Negative Values and Smoothness Regularization

Dijkstra's algorithm and its descendants can not plan with negative cost cells. Our version of Dijkstra's algorithm will expand a negative cost cell once to prevent infinite cycles. The planner may return an incorrect solution, forcing the gradients to add cost to the incorrectly visited cells. This naturally removes negative values, but only for the cells in the map that have been visited.

Ratliff *et al.* [20] suggests an optional projection step in the weight update to remove violations of any convex constraints. However, this approach does not seem tractable for a large non-linear network. We have also experimented with a regularization term to penalize negative cells directly

Algorithm 4 Psuedo-code for Max Margin Planning with Huber loss. w are network weights. $\mathcal{I}$ are map images. μ are expert paths of $|\mathcal{S}| \times |\mathcal{A}|$. Λ are language inputs. K is a spatial normalization constant, typically set to the map diagonal in pixels. Parameter δ sets Huber loss switch point.

```
function DeepMMPHuberLoss($w$,$\mathcal{I}$,$\mu$,$\Lambda$,$\delta$)
    $L \leftarrow 0$
    $dLdF \leftarrow 0$
    $F \leftarrow$Forward($w$,$\mathcal{I}$, $\mu$, $\Lambda$)                     ▷ Network forward pass
    for $F_i$, $\mu_i$ in $F$, $\mu$ do
        $len \leftarrow$PathLength($\mu_i$)
        $\beta_i \leftarrow \frac{K}{len}$
        $A \leftarrow F_i + \ell(\mu_i)$                                  ▷ Loss-augmented cost map
        $\hat{\mu}_i \leftarrow$Dijkstra($A$, $\mu_i$)              ▷ Cheapest path with same endpoints as expert
        $\mu_i^* \leftarrow$PathVisitation($\hat{\mu}_i$)           ▷ Convert to vectorized state visitation count
        $\mu_i \leftarrow$PathVisitation($\mu_i$)
        $L_x \leftarrow \beta_i \left\|\mathrm{vec}(F_i)^T \mu_i - \mathrm{vec}(A)^T \mu_i^*\right\|$
        if $|L_x| > \delta$ then
            $L \leftarrow L + \delta L_x - \frac{1}{2}\delta^2$                    ▷ $\mathrm{L}_1$ loss and gradient
            $dLdF \leftarrow dLdF + \delta\beta_i\,\mathrm{sgn}(\mathrm{vec}(F_i))(\mu_i - \mu_i^*)$
        else
            $L \leftarrow L + \frac{1}{2}(L_x)^2$                          ▷ $\mathrm{L}_2$ loss and gradient
            $dLdF \leftarrow dLdF + \beta_i L_x(\mu_i - \mu_i^*)$
        end if
    end for
return $\frac{L}{|\mathcal{I}|}$, $\frac{dLdF}{|\mathcal{I}|}$
end function
```

but have had the best results with the square or absolute value output functions (not shown in Equation (4.6)). Operations such as softmax do not work well because they are highly non-linear by design. In practice, we prefer the absolute value function since it does not amplify large values, especially true when $q = 2$ in Equation (4.6). We explore the selection of the output function under more realistic conditions in Section 6.1.2.

Finn *et al.* [65] successfully used Deep Inverse Optimal Control to learn trajectories for a robot arm. They use a path smoothness penalty expressed as the accumulation of a time derivative over the ground-truth path (Equation (4.12)), where $C(x)$ is the cost for state/action x. We have modified this equation to normalize the path length, which varies significantly between examples.

$$g_{\mathrm{lcr}} = \frac{\lambda_{\mathrm{lcr}}}{2} \sum_{x_t \in \mu_{ij}} \left[\frac{1}{|\mu_{ij}|} (C(x_{t+1}) - C(x_t)) - (C(x_t) - C(x_{t-1})) \right]^2 \tag{4.12}$$

This regularization did not help appreciably with our early experiments but may be worth revisiting with a more refined architecture.

4.3 Summary

This chapter detailed some of the fundamental work required to make Max Margin Planning a stable loss function for training a deep network. Robotics can be described as the practical implementation of theory from many disciplines, suitably adapted and augmented to fit reality. While our work is still far from operating on a functional robot, it must operate on the same computational machinery. Numerical precision and floating-point artifacts have been long-studied but are easily overlooked.

We designed the simplified experiments described in this chapter to uncover potential numerical stability concerns before they confound the development of our deep planning algorithm. It is difficult to debug a deep network that fails to converge or even diverges. The careful application of compensated summation allows 32-bit floating-point numbers to act with much higher precision. It is often challenging to fully utilize the powerful floating-point units on modern CPUs and GPUs. The addition of a few additional operations and registers costs very little. While CPUs can perform almost as well with double-precision floating-point, GPUs are especially weak in this regard, except for high-end scientific computing engines. Therefore, compensated summation is necessary for porting Max Margin Planning to the GPU.

We find that the initial state of an untrained network may lead to extreme losses. While the problem outlined above stems from adding many very small numbers, this problem is a consequence of exceedingly large ones. By using Huber Loss, we can operate in a mode where the Max Margin Planning Loss naturally constrains the loss gradient so that the training process remains stable. Unlike Ratliff *et al.* [20], we do not use a projection step to correct weight updates that violate planning constraints (i.e., negative values). In Chapter 5, we will substitute the linear conversion of features into a cost map with a highly non-linear deep network, and it is, therefore, more intuitive to use a non-negative activation as the output of the network instead.

Chapter 5

Deep Planning with Field D*

This chapter describes a few of the many specialized tools we developed to make this project practical. The recent wave of deep learning research over the last ten years would not have been possible without accessible tools and accelerated neural network building blocks. Toolboxes such as PyTorch [102], TensorFlow [103], Caffe [104], and others accelerate research by utilizing massively parallel computation without requiring in-depth knowledge of the underlying computational architecture.

While the work in this thesis makes great use of PyTorch (and initially Caffe), many of the more novel elements of our design are distant enough from commonly researched topics that there are no comparable frameworks or modules for accelerated computing. Our implementation of Field D* and the Fréchet coupling distance are generic enough that they are a useful contribution to others performing similar research in path planning. As these two algorithms are the major performance limiters for our research, we will detail their implementation here.

We begin with a comprehensive description of our implementation of Field D* for the CPU in Section 5.1. This is a re-statement of and expansion upon the work of Ferguson *et al.* [105], which we do for three important reasons.

First, it is essential to understand the optimization it performs so that we can modify it to produce gradients suitable for Max Margin Planning. Second, while the basic algorithm is stated in Ferguson *et al.* [105]–[107], many details of the path cost derivation are omitted. We show our version of the derivation in Section 5.2 to be explicit about our implementation, particularly in the case of path reconstruction and path termini that are not aligned to the cost field[1]. Finally, as planning is a bottleneck in our system, it is essential to understand the algorithm so that it performs well on both CPUs and GPUs.

We perform an analysis to justify the time and effort required to make a GPU accelerated version of the algorithm in Section 5.3, followed by technical details of our implementation in Section 5.3.1. Finally, we present an accelerated Fréchet coupling metric used for measuring planning performance.

We have also implemented the complete Max Margin Planning loss on the GPU. It is a straight-

[1] Additional details may be in the Ferguson *et al.* [107], but we have been unable to view a copy.

forward translation of the CPU version of the algorithm. The loss computation is a minor expense once the accelerated planner has computed the antecedent data; there is no need to redesign the algorithm to extract more parallelism. A GPU implementation eliminates the latency of transferring intermediate cost maps to the CPU and subsequently moving gradients to the GPU for back-propagation.

5.1 The Field D* Algorithm

Field D* (Ferguson *et al.* [107]) is a recognized and frequently used algorithm for field robot path planning. To our knowledge, it has not previously been used in the context of deep learning as part of a differentiable loss. There are two advantages of Field D* over Dijkstra's algorithm, used by Ratliff [58]. First, it is an any-angle path planner that generates more realistic paths without the restriction of having to transition between the 8-connected nearest neighbors on a grid. These trajectories more closely approximate what a human would generate, an important quality as our objective is to imitate human behavior. Second, Field D* can recompute paths when local costs change without recomputing the entire solution.

Field D* is a derivative of D*-Lite (Koenig *et al.* [108]) and Incremental A* (Koenig *et al.* [109]) and traces its lineage to Focused D* (Stentz [110]), D* (Stentz [111]), and eventually A* (Hart *et al.* [112]), which was used on Shakey the Robot (Nilsson [4]). All are solutions to variants of the Single Source Shortest Path (SSSP) problem. Similar to Dijkstra's algorithm, these algorithms frame the path planning problem as an incremental construction of $g(s)$, the lowest cost path from the state s to the goal[2].

$$g(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ \min_{s' \in Pred(s)} g(s') + c(s, s') & \text{otherwise} \end{cases} \tag{5.1}$$

The planners have a similar dynamic program that expands the frontier for efficient search (Equation (5.1)). The function $g(s)$ is the current estimated minimum cost from state s to the goal and $Pred(s)$ are the immediate predecessor states of s, with $c(s, s')$ the incremental cost of traveling between s and s'.

These planners are sequential and use a heuristic to select a node on the frontier to expand at each iteration. The heuristic biases the search to candidates likely to be on or near the path to the start vertex and often reduces the number of states visited before termination. However, this is not required for the algorithm to converge to the correct solution. This heuristic function, $h(s_1, s_2)$, is deemed admissible when it meets the condition in Equations (5.2) and (5.3).

$$h(s_{start}, s_{start}) = 0 \tag{5.2}$$

$$h(s, s_{start}) \leq c(s, s') + h(s', s_{start}) \tag{5.3}$$

[2] Different implementations of these algorithms begin the search from the goal or start vertex. We choose $g(s)$ to be the cost-to-goal following the convention of Field D*.

for all $s \in S$ and $s' \in Succ(s)$ where $s \neq s_{goal}$, and $Succ(s)$ is the set of permissible transitions to successor states. In short, the heuristic must always over-estimate the remaining cost to the goal compared to the true cost.

Modifications for Planning with Deep Networks

While the path planner used in this work is derived from Field D*, it also *not* a direct implementation of Field D*. The search heuristic has been removed because it is impossible to guarantee that the it always overestimates the cost of completing the path. The output function of our deep network is always non-negative, but there are no restrictions on the scale of the cost map or the minimum cost of any cell. Further, we eliminate the incremental replanning capability of Field D* since the entire cost map changes after each update pass of the deep network[3].

Without the A* search heuristic, the value of $g(s)$ is computed similarly to Dijkstra's algorithm, using the recurrence relation in Equation (5.1) but with a different traversal cost, to be introduced in Section 5.1.1. Once the algorithm terminates, the shortest path is constructed from the start location, choosing the next state with Equation (5.4) at each step until the goal is reached.

$$s_{i+1} = \underset{s' \in Pred(s_i)}{\operatorname{argmin}} \; g(s') \tag{5.4}$$

The planner presented here is best viewed as a modified version of Dijkstra's algorithm that has been augmented with the cost interpolation technique from Field D*. Likewise, our GPU-accelerated version (Section 5.3.1) is best viewed as a hybrid of the Bellman-Ford algorithm ([113], [114]) with the same interpolation technique borrowed from Field D*. We refer to our implementation as a version Field D* to highlight its lineage.

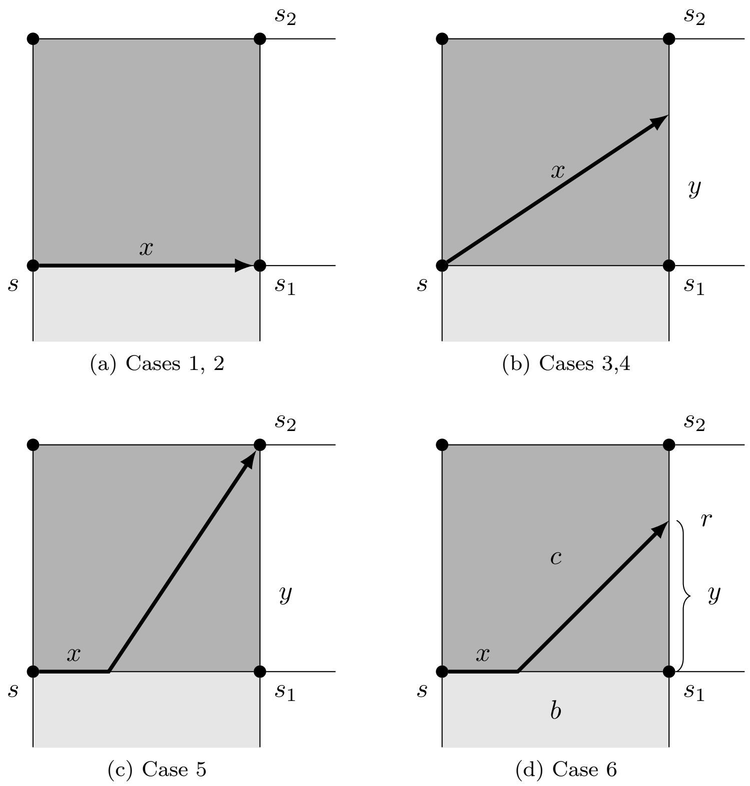
5.1.1 Field D* Cost Derivation

In this section, we restate and analyze the field cost computation from Ferguson *et al.* [105]. We do this primarily to understand how to generate gradients suitable for Max Margin Planning. In addition, we must implement it efficiently for CPUs and GPUs after removing parts not relevant in our context.

While the basic algorithm is stated in Ferguson *et al.* [105]–[107], many details of the path reconstruction are omitted. The authors note that Field D* is typically used to calculate a global trajectory in conjunction with an arc-based local planner, possibly explaining why this information was not published. We show our version of the derivation to be explicit about our implementation of the cost interpolation algorithm, particularly in the case of path reconstruction. During our analysis, we have discovered an inconsistency in the interpolated cost computation, which we address in Section 5.1.3.

Field D* allows paths to transition between cells less restrictively than the 8-connected neighbors visited by A*. Instead of computing the cost of travel from state to state, Field D* computes the

[3] Future work may utilize this feature, updating only a part of the cost map as a robot explores.



(a) Cases 1, 2

(b) Cases 3,4

(c) Case 5

(d) Case 6

Figure 5.1: Traversal cases for Field D* path. Figure adapted from Ferguson *et al.* [107]. Values b and c are the traversal costs for the cell. Black dots are values of field $g(s)$.

traversal cost at discrete field points that are interpolated to represent the cost of paths that may not align to the grid (Figure 5.1). Unlike A* and its variants, the field points, s, are located at the junctions between map cells.

When the path is on the border of two cells, the incremental cost for a path segment $\overrightarrow{ss_1}$ is defined by Equation (5.5), illustrated in Figure 5.1a.

$$|g(s_1) - g(s)| = \min(b, c) \tag{5.5}$$

The estimated onward cost of any path starting on the edge $\overrightarrow{s_1s_2}$ is a linear interpolation of the field values $g(s_1)$ and $g(s_2)$ (Equation (5.6)).

$$g(s_y) = g(s_1)y + g(s_2)(1 - y) \tag{5.6}$$

Combining the two equations leads to the general form of the recursive cost for a Field D* path

(Equation (5.7)).

$$g(s) = \min_{x,y} \left[bx + c\sqrt{(1-x)^2 + y^2} + yg(s_2) + (1-y)g(s_1) \right] \tag{5.7}$$

The shape of the path is determined by the parameters $x \in [0,1]$ and $y \in [0,1]$. The cell traversal cost is a linear combination of the fixed map costs b and c and the free variables x and y. By selecting appropriate x and y, Field D* can generate the paths in Figure 5.1 and Table 5.1, with the exception of a path where $x > 0$ and $y > 0$ (Case 6 in Table 5.1), which is explictly forbidden because it is never optimal. The reason for this is discussed in Section 5.1.4.

Table 5.1: Cell traversal cases for varying x and y parameters. Case 6 is never optimal. See Section 5.1.4 for explanation.

Case	Figure	x	y	Path Description
1	Figure 5.1a	0	0	Boundary between b and c to s_1
2	Figure 5.1a	1	0	Boundary between b and c to s_1
3	Figure 5.1b	0	1	Diagonal across cell to s_2
4	Figure 5.1b	0	$0 < y < 1$	Diagonal across cell, does not terminate at corner
5	Figure 5.1c	$0 < x < 1$	1	Part of bottom, then direct to s_2
6	Figure 5.3	$0 < x < 1$	$0 < y < 1$	Part of the bottom, part of the right edge of cell, may or may not terminate at corner

For convenience, Ferguson *et al.* [107] rewrite Equation (5.7) by substituting $f = g(s_1) - g(s_2)$ to yield Equation (5.8).

$$g(s) = \min_{x,y} \left[bx + c\sqrt{(1-x)^2 + y^2} + (1-y)f + g(s_2) \right] \tag{5.8}$$

The variable f is a decision variable that features prominently in the optimization strategy for the incremental path cost in Algorithm 5, as reproduced from Ferguson *et al.* [107]. We now inspect the cases in Table 5.1 and Algorithm 5 in more detail.

5.1.2 Cases 1 & 2

If $f < 0$, then Ferguson *et al.* [107] state without a detailed proof that it is always cheapest to go direct to s_1 (Figure 5.1a). The path cost is defined by Equation (5.11) when $y = 0$ and $x \in 0,1$ and is reflected in Algorithm 5 line 7 and line 9. Our re-implementation of Field D* has revealed that this is not always the case.

$$g(s) = \min_{x,y} \left[bx + c\sqrt{(1-x)^2 + 0^2} + (1-0)f + g(s_2) \right] \qquad y = 0 \tag{5.9}$$

$$g(s) = \min_{x,y} \left[bx + c\sqrt{(1-x)^2} + g(s_1) - g(s_2) + g(s_2) \right] \qquad x \in 0,1 \tag{5.10}$$

$$g(s) = \min(b,c) + g(s_1) \tag{5.11}$$

Algorithm 5 Cost computation from Ferguson *et al.* [105]. This version has an error described in Section 5.1.3.

```
function BasicCost($g(s_1)$, $g(s_2)$, $c$, $b$)
    if $\min(b, c) = \infty$ then
        $vs \leftarrow \infty$
    else
        if $g(s_1) < g(s_2)$ then
            if $b < c$ then                                          ▷ Case 2
                $x \leftarrow 1$   $y \leftarrow 0$
            else                                                     ▷ Case 1
                $x \leftarrow 0$   $y \leftarrow 0$
            end if
            $vs \leftarrow \min(b, c) + g(s_1)$
        else
            $f = g(s_1) - g(s_2)$
            if $f \leq b$ then
                if $c \leq f$ then                                   ▷ Case 3
                    $x \leftarrow 0$   $y \leftarrow 1$
                    $vs \leftarrow c\sqrt{2} + g(s_2)$
                else                                                 ▷ Case 4
                    $x \leftarrow 0$
                    $y \leftarrow \min(1, \frac{f}{\sqrt{c^2 - f^2}})$
                    $vs \leftarrow c\sqrt{1 + y^2} + f(1 - y) + g(s_2)$
                end if
            else
                if $c \leq b$ then                                   ▷ Case 3
                    $x \leftarrow 0$   $y \leftarrow 1$
                    $vs \leftarrow c\sqrt{2} + g(s_2)$
                else                                                 ▷ Case 5
                    $x \leftarrow 1 - \min(1, \frac{b}{\sqrt{c^2 - b^2}})$   $y \leftarrow 1$
                    $vs \leftarrow bx + c\sqrt{1 + (1 - x)^2} + g(s_2)$
                end if
            end if
        end if
    end if
    return $vs$, $x$, $y$
end function
```

Consider Equation (5.12), which must be true when $f < 0$, as asserted in Algorithm 5. It states that a direct path to s_1 must *always* be cheaper than a diagonal path to some intermediate state between s_1 and s_2.

$$\min(b, c) + g(s_1) \leq c\sqrt{1 + y^2} + f(1 - y) + g(s_2) \tag{5.12}$$

$$\min(b, c) \leq c\sqrt{1 + y^2} + f(1 - y) - (g(s_1) - g(s_2)) \tag{5.13}$$

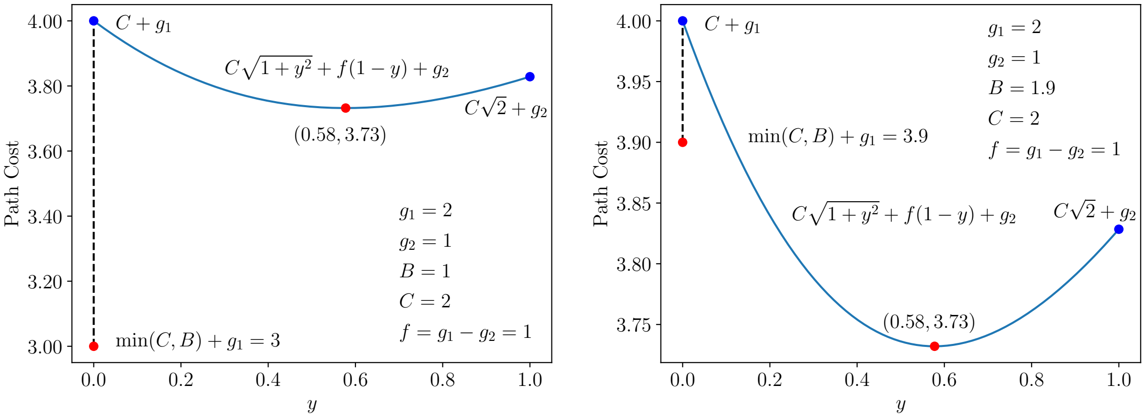
$$\min(b, c) \leq c\sqrt{1 + y^2} + f(1 - y) - f \tag{5.14}$$

$$\min(b, c) \leq c\sqrt{1+y^2} - fy \tag{5.15}$$

It is clear that $0 < y < 1$ is valid solution for certain $b > 0$ and $c > 0$ *even* when $f < 0$. Cases 1 and 2 may be optimal even when $f > 0$.

5.1.3 Counterexample

Consider an example where $g(s_1) = 2$, $g(s_2) = 1$, $c = 2$ and $b = 1$. Since $g(s_1) \geq g(s_2)$, Algorithm 5 will execute case 4 (Line 19) and return a value of 3.73 (Figure 5.2a). However, case 2 is even cheaper (Line 7), traversing the bottom edge of the cell with cost $\min(b, c) + g(s_1) = 1 + 2 = 3$. The real solution to the boundary condition in eq. (5.12) lays along a quadratic, without consideration of the case where $x \neq 0$.



(a) Incorrect

(b) Correct

Figure 5.2: Algorithm 5 returns the incorrect minimum in Figure 5.2a. Figure 5.2b is an example of a correct result from Algorithm 5, after adjusting B.

5.1.4 Excluding Case 6

Case 6 in Table 5.1 is a three segment path, where $x \in (0, 1]$ and $y \in (0, 1]$ which is never optimal. Ferguson *et al.* [107] construct a geometric proof to show this (Figure 5.3), reproduced in greater detail here. Suppose the optimal path passes through $\overrightarrow{ss_x s_y s_2}$. Then the cost of this path must be less than the cost around the perimeter of the cell (Equation (5.16)).

$$x\min(b, c) + c\sqrt{(1-x)^2 + y^2} + (1-y)f \leq x\min(b, c) + (1-x)\min(b, c) + yf + (1-y)f \tag{5.16}$$

$$c\sqrt{(1-x)^2 + y^2} \leq (1-x)\min(b, c) + yf \tag{5.17}$$

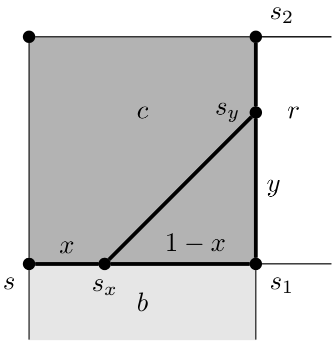


Figure 5.3: Setting for geometric proof that a path where both $x \in (0, 1]$ and $y \in (0, 1]$ is sub-optimal.

Equation (5.17) is an inequality on the sides of a right triangle. Suppose we carefully choose $y_2 = (1 - y)$ and $x_2 = (1 - x)(1 - y)/y$, which yields an inequality for a similar triangle (Equation (5.18)).

$$c\sqrt{(1 - x_2)^2 + y_2^2} \leq (1 - x_2) \min(b, c) + y_2 f \tag{5.18}$$

Because the triangles are similar, we can add the triangle sides to yield a new path. We compute the cost by adjusting the cost of $g(s)$, the cost for the original path (Equation (5.19)).

$$g(s)' = g(s) \overbrace{- \underbrace{(1 - x_2) \min(b, c)}_{\text{Shortened } \overrightarrow{ss_x}} + \underbrace{c\sqrt{(1 - x_2)^2 + (1 - y)^2}}_{\text{Lengthened hypotenuse}} - \underbrace{(1 - y)f}_{\text{Shortened } \overrightarrow{s_y s_2}}}^{\text{K}} \tag{5.19}$$

However, from Equation (5.18) we know:

$$\sqrt{(1 - x_2)^2 + (1 - y)^2} \leq (1 - x_2) \min(b, c) + (1 - y)f \tag{5.20}$$

$$K = -(1 - x_2) \min(b, c) + \sqrt{(1 - x_2)^2 + (1 - y)^2} - (1 - y)f \leq 0 \tag{5.21}$$

We substitute Equation (5.21) into Equation (5.19) to show that $g(s)' \leq g(s)$, and $g(s)$ is never optimal for $x \in (0, 1]$, $y \in [0, 1)$.

5.1.5 General Case

The general formula for the cost is:

$$g(s) = bx + c\sqrt{(1 - x)^2 + y^2} + f(1 - y) + g(s_2) \tag{5.22}$$

We have excluded the possibility that both $x > 0$ and $y > 0$ in Section 5.1.4. If $f < b$, then it is cheaper to optimize y and set $x = 0$. If $f > b$, then it is cheaper to optimize x and set $y = 0$. To understand this, consider the inequality in Equation (5.23). If $f < b$, then:

$$c\sqrt{1+y^2} + f(1-y) + g_2 \leq bx + c\sqrt{(1-x)^2+1} + g_2 \tag{5.23}$$

$$c\sqrt{1+y^2} + f(1-y) \leq bx + c\sqrt{(1-x)^2+1} \tag{5.24}$$

Substituting $x = 1 - y'$, Equation (5.26) is only true if $f \leq b$ and it is therefore better to optimize y instead of y'.

$$c\sqrt{1+y^2} + f(1-y) \leq b(1-y') + c\sqrt{(1-(1-y'))^2+1} \tag{5.25}$$

$$c\sqrt{1+y^2} + f(1-y) \leq b(1-y') + c\sqrt{y'^2+1} \tag{5.26}$$

We now optimize the two cases, first supposing $x = 0$, then:

$$g(s) = c\sqrt{1+y^2} + f(1-y) + g(s_2) \tag{5.27}$$

Take derivative and set it to zero to find optimal y^*.

$$\frac{\partial g(s)}{\partial y} = \frac{1}{2}c(1+y^2)^{-\frac{1}{2}}2y - f = 0 \tag{5.28}$$

$$\frac{cy}{\sqrt{(1+y^2)}} = f \tag{5.29}$$

$$\frac{c^2y^2}{1+y^2} = f^2 \tag{5.30}$$

$$c^2y^2 = f^2 + f^2y^2 \tag{5.31}$$

$$y^* = \sqrt{\frac{f^2}{c^2-f^2}} \tag{5.32}$$

Clearly, y^* can be arbitrarily large but can not be larger than 1, the cell size.

$$y^* = \sqrt{\frac{f^2}{c^2-f^2}} \leq 1 \tag{5.33}$$

$$\frac{f^2}{c^2-f^2} \leq 1 \tag{5.34}$$

$$f^2 \leq c^2 - f^2 \tag{5.35}$$

$$c \geq f\sqrt{2} \tag{5.36}$$

This bound is more strict than $c > f$ in Algorithm 5, which is required to prevent the denominator of Equation (5.32) and Line 20 from becoming zero or negative. Note that the algorithm also limits $y \leq 0$.

Suppose $y = 1$, then:

$$g(s) = bx + c\sqrt{(1-x)^2+1} + g(s_2) \tag{5.37}$$

Take the derivative and set it to zero to find optimal x^*.

$$\frac{\partial g(s)}{\partial x} = b + \frac{1}{2}c\left((1-x)^2+1\right)^{-\frac{1}{2}} 2(1-x)(-1) = 0 \tag{5.38}$$

$$b = c\frac{1-x}{\sqrt{(1-x)^2+1}} \tag{5.39}$$

$$x^* = 1 - \frac{b}{\sqrt{c^2-b^2}} \tag{5.40}$$

We must ensure that $x \geq 0$.

$$0 \leq 1 - \frac{b}{\sqrt{c^2-b^2}} \tag{5.41}$$

$$c^2 - b^2 \geq b^2 \tag{5.42}$$

$$c \geq b\sqrt{2} \tag{5.43}$$

5.1.6 Revised Algorithm

The optimization process is summarized in Equation (5.44) and our revised Algorithm 6, without the decision variables used in Algorithm 5.

$$g(s) = \min \begin{cases} \min(b,c) + g(s_1) & \\ b(\underbrace{1 - \frac{b}{\sqrt{c^2-b^2}}}_{x^*}) + c\sqrt{1+\frac{b^2}{c^2-b^2}} + g(s_2) & \text{for } c \geq b\sqrt{2} \\ f(\underbrace{1 - \frac{f}{\sqrt{c^2-f^2}}}_{y^*}) + c\sqrt{1+\frac{f^2}{c^2-f^2}} + g(s_2) & \text{for } c \geq f\sqrt{2} \\ c\sqrt{2} + g(s_2) & \end{cases} \tag{5.44}$$

The solution cases' criteria are almost as complex as the solutions themselves. These criteria become even more complex if the map cells are not unit-sized, which happens when reconstructing the optimal path. Computing the minimum of all solutions naturally corrects the erroneous case in Section 5.1.3 and replaces a control-flow dependency in the program with a data-flow dependency. Instead of branching to a different case of Equation (5.44), the CPU can compute all solutions and choose the minimum in a single stream of mathematical operations. This technique better utilizes the pipelined floating-point units in modern CPUs and is especially critical to achieving good performance with a GPU.

5.1.7 Computing the Field

As with Dijkstra's algorithm, Field D* is fundamentally a sequential breadth-first search that expands the state with the current cheapest path to the start state. States that are not yet visited

Algorithm 6 Cost computation from Ferguson *et al.* [105] with correction from Section 5.1.3. The min function returns the tuple with the smallest first element.

```
function BasicCostV1($g(s_1)$, $g(s_2)$, $c$, $b$)
    if $\min(b, c) = \infty$ then
        $vs \leftarrow \infty \quad x \leftarrow 0 \quad y \leftarrow 0$
    else
        $f \leftarrow g(s_1) - g(s_2)$
        if $(b < c)$ then $x_1 \leftarrow 1$ else $x_1 \leftarrow 0$
        $v_1 \leftarrow \min(b, c) + g(s_1)$                              ▷ Cases 1,2
        $v_3 \leftarrow c\sqrt{2} + g(s_2)$                                ▷ Case 3
        $v_4 \leftarrow \infty \quad y_4 \leftarrow 0$
        $v_5 \leftarrow \infty \quad x_5 \leftarrow 0$
        $t \leftarrow c^2 - f^2$
        if $t > 0$ then
            $y_4 \leftarrow \frac{f}{\sqrt{t}}$
            if $y_4 \leq 1$ then
                $v_4 \leftarrow c\sqrt{1 + y_4^2} + f(1 - y_4) + g(s_2)$   ▷ Case 4
            end if
        end if
        if $c > b$ then
            $x_5 \leftarrow 1 - \frac{b}{\sqrt{c^2 - b^2}}$
            if $x_5 > 0$ then
                $v_5 \leftarrow bx_5 + c\sqrt{(1 - x_5)^2 + 1} + g(s_2)$   ▷ Case 5
            end if
        end if
        $vs, x, y \leftarrow \min\left((v_1, x_1, 0),\ (v_3, 0, 1),\ (v_4, 0, y_4),\ (v_5, x_5, 1)\right)$
    end if
    return $vs, x, y$
end function
```

are given a value of infinity. Once a frontier state is selected, the field is updated using the cost map and the g value of neighboring nodes, as described by Algorithm 7.

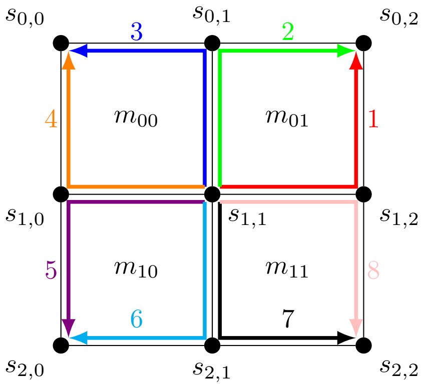
Algorithm 7 Computation of single field element, $g(s)$.

```
function FieldVertexCost($g$,$m$)
    $v_1 \leftarrow$ BasicCostV1$(g(s_{12}), g(s_{02}), m(s_{01}), m(s_{11}))$
    $v_2 \leftarrow$ BasicCostV1$(g(s_{01}), g(s_{02}), m(s_{01}), m(s_{00}))$
    $v_3 \leftarrow$ BasicCostV1$(g(s_{01}), g(s_{00}), m(s_{00}), m(s_{01}))$
    $v_4 \leftarrow$ BasicCostV1$(g(s_{10}), g(s_{00}), m(s_{00}), m(s_{10}))$
    $v_5 \leftarrow$ BasicCostV1$(g(s_{10}), g(s_{20}), m(s_{10}), m(s_{00}))$
    $v_6 \leftarrow$ BasicCostV1$(g(s_{21}), g(s_{20}), m(s_{10}), m(s_{11}))$
    $v_7 \leftarrow$ BasicCostV1$(g(s_{21}), g(s_{22}), m(s_{11}), m(s_{10}))$
    $v_8 \leftarrow$ BasicCostV1$(g(s_{12}), g(s_{22}), m(s_{11}), m(s_{01}))$
    return $\min(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_7)$
end function
```

We use matrix notation to identify the neighboring map and field values, as detailed in Figure 5.4.

Case	s_1	s_2	c	b
1	s_{12}	s_{02}	m_{01}	m_{11}
2	s_{01}	s_{02}	m_{01}	m_{00}
3	s_{01}	s_{00}	m_{00}	m_{01}
4	s_{10}	s_{00}	m_{00}	m_{10}
5	s_{10}	s_{20}	m_{10}	m_{00}
6	s_{21}	s_{20}	m_{10}	m_{11}
7	s_{21}	s_{22}	m_{11}	m_{10}
8	s_{12}	s_{22}	m_{11}	m_{01}

(a) Parameters for each optimization in Algorithm 7.



(b) Potential solutions for minimal cost-to-goal path.

Figure 5.4: Field D* computes the minimal cost-to-goal for a path originating at s_{11} through any of eight neighbor edges.

A state, s_{ij}, is located to the upper left of a map cell m_{ij}. When computing path costs, we assume that vertex $s = s_{11}$. The colored arrows in Figure 5.4b depict the different solutions in Algorithm 7.

5.1.8 Terminating Conditions

The original Field D* algorithm terminates when the start vertex is removed from the heap. Because we start and end on nodes in the middle of map cells, we instead terminate when the four corner vertices of the starting map cell are marked closed and are self-consistent. With a field value for all edges of the cell, we can use the path extraction algorithm to interpolate a path from the start to the goal.

5.2 Path Extraction

Path extraction requires a more general cost computation formula than used for estimating the field. This derivation was not found in Ferguson *et al.* [105]–[107], but is based on the principles presented in those papers and outlined in Section 5.1. Consider a path at the edge of a map cell with $0 < y < 1$ after the prior extraction step. This path will end on a horizontal or vertical edge (Figure 5.5), and we need to find the next waypoint to the goal.

This is a general case of the earlier computation, with cells that are not unit sized, having been scaled by s_x and s_y, where $s_x, s_y \in (0, 1]$. The traversal cost is defined by Equation (5.45).

$$g(s) = bs_x x + c\sqrt{(s_x(1-x))^2 + (s_y y)^2} + f(1-y) + g(s_2) \tag{5.45}$$

We do not yet consider the case when both $s_x < 1$ and $s_y < 1$. This will be addressed in Section 5.2.1.

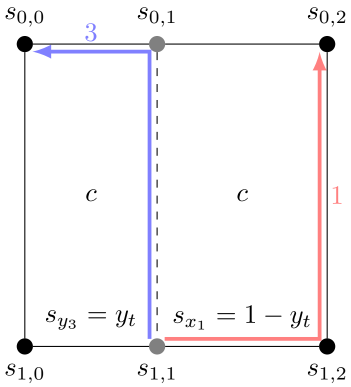


Figure 5.5: Cost interpolation for point on map cell boundary. Paths 1 and 3, labeled with same convention as Figure 5.4b, have scaling factors s_{x_1}, s_{y_3} for the x and y coordinates respectively.

Note that f is also scaled. For path 3, f is defined in Equation (5.46).

$$f = g(s_{0,1}) - g(s_{0,0}) \tag{5.46}$$

$$g(s_{0,1}) = g(s_{0,0}) + s_{y_3}\left(g(s_{0,2}) - g(s_{0,0})\right) \tag{5.47}$$

Suppose $x = 0$, then:

$$g(s) = c\sqrt{s_x^2 + s_y^2 y^2} + f(1-y) + g(s_2) \tag{5.48}$$

Take derivative and set it to zero to find optimal y^*.

$$\frac{\partial g(s)}{\partial y} = \frac{1}{2}c(s_x^2 + s_y^2 y^2)^{-\frac{1}{2}} 2 s_y^2 y - f = 0 \tag{5.49}$$

$$\frac{c s_y^2 y}{\sqrt{(s_x^2 + s_y^2 y^2)}} = f \tag{5.50}$$

$$(c s_y^2 y)^2 - (f s_y y)^2 = (f s_x)^2 \tag{5.51}$$

$$s_y^2 y^2 (s_y^2 c^2 - f^2) = (f s_x)^2 \tag{5.52}$$

$$y^* = \sqrt{\frac{(f s_x)^2}{s_y^2((s_y c)^2 - f^2)}} = \frac{s_x}{s_y}\frac{f}{\sqrt{(s_y c)^2 - f^2}} \tag{5.53}$$

We must ensure that $y^* \leq 1$, which happens when Equation (5.59) is satisfied.

$$y^* = \frac{s_x}{s_y}\sqrt{\frac{f^2}{(s_y c)^2 - f^2}} \leq 1 \tag{5.54}$$

$$f^2 \leq \left(\frac{s_y}{s_x}\right)^2 ((s_y c)^2 - f^2) \tag{5.55}$$

$$f^2 \left(\frac{s_x}{s_y}\right)^2 (1 + \left(\frac{s_y}{s_x}\right)^2) \leq (s_y c)^2 \tag{5.56}$$

$$\sqrt{\frac{f^2}{s_y^2}\left(\left(\frac{s_x}{s_y}\right)^2+1\right)} \leq c \tag{5.57}$$

$$\sqrt{\frac{f^2}{s_y^2}\left(\left(\frac{s_x}{s_y}\right)^2+\left(\frac{s_y}{s_y}\right)^2\right)} \leq c \tag{5.58}$$

$$c \geq \frac{f}{s_y^2}\sqrt{s_x^2+s_y^2} \tag{5.59}$$

Suppose $y = 1$, then the cost is defined by Equation (5.60).

$$g(s) = bs_x x + c\sqrt{(s_x(1-x))^2+s_y^2} + g(s_2) \tag{5.60}$$

Take derivative and set it to zero to find optimal x^*.

$$\frac{\partial g(s)}{\partial x} = bs_x + \frac{1}{2}c((s_x(1-x))^2+s_y^2)^{-\frac{1}{2}}2s_x^2(1-x)(-1) = 0 \tag{5.61}$$

$$bs_x = c\frac{s_x^2(1-x)}{\sqrt{(s_x(1-x))^2+s_y^2}} \tag{5.62}$$

$$(bs_x s_y)^2 = (c^2-b^2)s_x^4(1-x)^2 \tag{5.63}$$

$$\frac{(bs_x s_y)^2}{c^2-b^2} = s_x^4(1-x)^2 \tag{5.64}$$

$$\frac{bs_y}{s_x\sqrt{c^2-b^2}} = (1-x) \tag{5.65}$$

$$x^* = 1 - \frac{s_y}{s_x}\frac{b}{\sqrt{c^2-b^2}} \tag{5.66}$$

We must ensure that $x \geq 0$, which happens when Equation (5.70) is satisfied.

$$0 \leq 1 - \frac{s_y}{s_x}\frac{b}{\sqrt{c^2-b^2}} \tag{5.67}$$

$$1 \geq \left(\frac{s_y}{s_x}\right)^2\frac{b^2}{c^2-b^2} \tag{5.68}$$

$$c^2-b^2 \geq \left(\frac{s_y}{s_x}\right)^2 b^2 \tag{5.69}$$

$$c \geq b\sqrt{1+\left(\frac{s_y}{s_x}\right)^2} = \frac{b\sqrt{s_x^2+s_y^2}}{s_x} \tag{5.70}$$

It is no longer simple to determine from inspection if the optimal solution has $x > 0$ or $y > 0$ as in Equation (5.23). We again construct the cost inequality, and substitute $x = 1 - y'$.

$$bs_x x + c\sqrt{(s_x(1-x))^2+s_y^2} + g(s_2) \geq c\sqrt{s_x^2+(s_y y)^2} + f(1-y) + g(s_2) \tag{5.71}$$

$$bs_x x + c\sqrt{(s_x(1-x))^2 + s_y^2} \geq c\sqrt{s_x^2 + (s_y y)^2} + f(1-y) \tag{5.72}$$

$$bs_x(1-y') + c\sqrt{(s_x(1-(1-y')))^2 + s_y^2} \geq c\sqrt{s_x^2 + (s_y y)^2} + f(1-y) \tag{5.73}$$

$$bs_x(1-y') + c\sqrt{(s_x y')^2 + s_y^2} \geq c\sqrt{s_x^2 + (s_y y)^2} + f(1-y) \tag{5.74}$$

It does not seem as if a closed-form solution is possible because of the complex dependency on s_x and s_y when $s_x \neq s_y$. As before, we compute all real solutions and then select a minimum, as in Algorithm 8 and Equation (5.75). The boundaries between solutions are at least as complex to compute as the solutions themselves. Computing the minimum of all possible solutions is less error-prone, as well. This algorithm is used primarily during path reconstruction and does not contribute significantly to run-time.

$$g(s) = \min \begin{cases} \min(b,c)s_x + g(s_1) \\ bs_x \underbrace{\left(1 - \frac{s_y}{s_x}\frac{b}{\sqrt{c^2-b^2}}\right)}_{x^*} + c\sqrt{s_y^2 + \left(\frac{bs_y}{\sqrt{c^2-b^2}}\right)^2} + g(s_2) \\ c\sqrt{s_x^2 + \left(\frac{s_x f}{\sqrt{(s_y c)^2-f^2}}\right)^2} + f\left(1 - \underbrace{\frac{s_x}{s_y}\frac{f}{\sqrt{(s_y c)^2-f^2}}}_{y^*}\right) + g(s_2) \\ c\sqrt{s_x^2+s_y^2} + g(s_2) \end{cases} \tag{5.75}$$

The optimal $x^*, y^* \in [0,1]$ from these computations must be scaled by s_x or s_y and appropriately offset to recover solutions in the same coordinate space as the map.

5.2.1 Non-Aligned Path Endpoints

Our datasets use map-centered coordinates. Integer coordinates are aligned with the center of map cells, whereas $g(s)$ vertices are defined at the corners of map cells. This section adapts the cost interpolation to suit the case where a path endpoint is at the center of a map cell.

Field D* constructs $g(s)$, the cost to the goal from state s, as an iterative expansion from the goal point. If the goal point is not already on a field vertex, we must compute the cost from the field vertices that bracket the goal point. We term this the internal cost-to-goal computation, illustrated in Figure 5.6a and detailed in Section 5.2.2. The external cost-to-goal computation is used for paths that end on the edge of a map cell that contains a goal point and is illustrated in Figure 5.6b and described in Section 5.2.3.

5.2.2 Internal Cost-to-Goal

For each field vertex that brackets the goal point we compute the cost to the goal using the interpolated path equations in Algorithm 8 with the added constraints that all paths must end at

Algorithm 8 Interpolated cost computation.

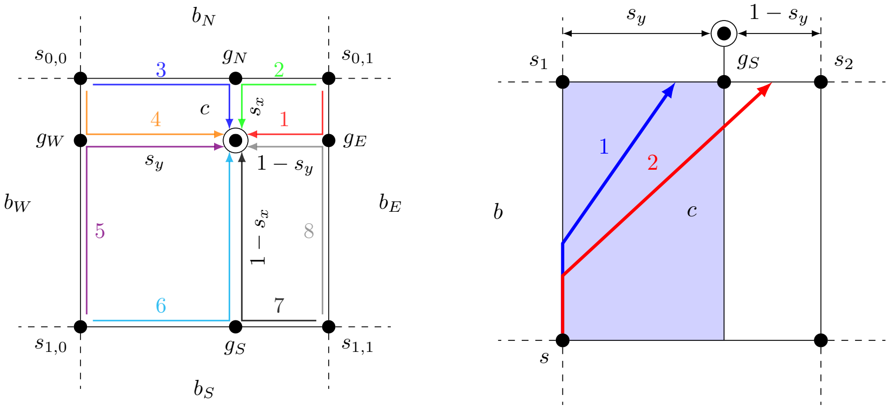
```
function INTERPCOSTV1($g(s_1), g(s_2), c, b, s_x, s_y$)
    if $\min(b, c) = \infty$ then
        $vs \leftarrow \infty \quad x \leftarrow 0 \quad y \leftarrow 0$
    else
        $f \leftarrow g(s_1) - g(s_2)$
        if $(b < c)$ then $x_1 \leftarrow 1$ else $x_1 \leftarrow 0$
        $c_1 \leftarrow \min(b, c)s_x + g(s_1)$
        $c_2 \leftarrow c\sqrt{s_x^2 + s_y^2} + g(s_2)$
        $c_3 \leftarrow \infty \quad y_3 \leftarrow 0$
        $c_4 \leftarrow \infty \quad x_3 \leftarrow 0$
        $t \leftarrow (s_y c)^2 - f^2$
        if $t > 0$ then
            $y_3 \leftarrow \frac{s_x}{s_y} \frac{f}{\sqrt{t}}$
            if $y \leq 1$ then
                $c_3 \leftarrow c\sqrt{s_x^2 + (s_y y_3)^2} + f(1 - y_3) + g(s_2)$
            end if
        end if
        if $c > b$ then
            $x_4 \leftarrow 1 - \frac{s_y}{s_x} \frac{b}{\sqrt{c^2 - b^2}}$
            if $x_4 > 0$ then
                $c_4 \leftarrow b s_x x_4 + c\sqrt{s_x^2(1 - x_4)^2 + s_y^2} + g(s_2)$
            end if
        end if
        $vs, x, y \leftarrow \min\left((c_1, x_1, 0),\ (c_2, 0, 1),\ (c_3, 0, y_3),\ (c_4, x_4, 1)\right)$
    end if
    return $vs, x, y$
end function
```

the goal, s_2. The goal flag is added to Algorithm 9 for this purpose. We fix $g(s_2) = 0$. Depending on the path taken in Figure 5.6a, $g(s_1)$ is one of g_N, g_S, g_E, g_W. The value of g_x represents the cost of the direct path to s_2 (Equation (5.76) and Algorithm 9).

$$g(s_1) = \begin{cases} g_N = cs_x \\ g_S = c(1 - s_x) \\ g_E = c(1 - s_y) \\ g_W = cs_y \end{cases} \tag{5.76}$$

5.2.3 External Cost-to-Goal

When updating the field value of a vertex (s in Figure 5.6b) with both s_1 and s_2 bracketing a goal cell, another special routine is required. This is because $g_S \neq (g(s_2) - g(s_1))s_y + g(s_1)$. The cost



(a) Internal cost-to-goal computation.

(b) External cost-to-goal computation.

Figure 5.6: Cost-to-goal computation for field estimation.

along edge $\overrightarrow{s_1 s_2}$ is piecewise linear and there are two potential solutions, with either $y \in [0, s_y]$ or $y \in [s_y, 1]$. For the second case, we must further generalize Algorithm 8 to enforce solutions that traverse the shaded area in Figure 5.6b but only return solutions $y \in [s_y, 1]$.

Starting with Equation (5.77), we insert an offset term y_o to account for the traversal of the shaded region in Figure 5.6b. When the offset term is non-zero, solutions that terminate on the interval $y \in [s_1, g_S)$ are prohibited.

$$g(s) = b s_x x + c\sqrt{(s_x(1-x))^2 + (s_y y + y_o)^2} + f(1-y) + g(s_2) \tag{5.77}$$

If $x = 0$, we can solve for y^* by taking the derivative of the cost and setting it to 0. The optimal y^* is expressed by Equation (5.79).

$$y^* = -\frac{f s_x \sqrt{c^2 {s_y}^2 - f^2} + f^2 y_o - c^2 {s_y}^2 y_o}{s_y(f^2 - c^2 {s_y}^2)} \tag{5.78}$$

$$y^* = \frac{f s_x}{s_y} \frac{1}{\sqrt{c^2 {s_y}^2 - f^2}} - \frac{y_o}{s_y} \tag{5.79}$$

For $y = 1$, the optimal value for x^* is expressed by Equation (5.81).

$$x^* = \frac{b^2 s_x - c^2 s_x + b s_y \sqrt{c^2 - b^2} + b y_o \sqrt{c^2 - b^2}}{s_x\,(b^2 - c^2)} \tag{5.80}$$

$$x^* = 1 - \frac{b(s_y + y_o)}{s_x \sqrt{c^2 - b^2}} \tag{5.81}$$

The new results are summarized in Equation (5.82) and realized in Algorithm 9, with appropriate tests to ensure a valid domain and range for each case.

With the core routine complete, we can write the update rule for field values when s_1 and s_2 are

Algorithm 9 Interpolated cost computation amended for cost-to-goal computation. Goal flag excludes cases that do not terminate at $s2$, which is located at the goal point (Figure 5.6a).

```
1: function InterpCostV2($g(s_1)$, $g(s_2)$, $c$, $b$, $s_x$, $s_y$, $y_o$, goal)
2:     if $s_x = 0$ and $s_y = 0$ then
3:         $vs \leftarrow 0 \quad x \leftarrow 0 \quad y \leftarrow 0$
4:     else
5:         if $\min(b, c) = \infty$ then
6:             $vs \leftarrow \infty \quad x \leftarrow 0 \quad y \leftarrow 0$
7:         else
8:             $f \leftarrow g(s_1) - g(s_2)$
9:             $c_1 \leftarrow \infty$
10:            $c_2 \leftarrow c\sqrt{s_x^2 + (s_y + y_o)^2} + g(s_2)$
11:            $c_3 \leftarrow \infty \quad y_3 \leftarrow 0$
12:            $c_4 \leftarrow \infty \quad x_4 \leftarrow 0$
13:            if goal = False then
14:                if $y_o = 0$ then
15:                    if $(b < c)$ then $x_1 \leftarrow 1$ else $x_1 \leftarrow 0$
16:                    $c_1 \leftarrow \min(b, c)s_x + g(s_1)$
17:                end if
18:                $t \leftarrow (s_y c)^2 - f^2$
19:                if $t > 0$ and $s_y > 0$ then
20:                    $y_3 \leftarrow \frac{s_x}{s_y}\frac{f}{\sqrt{t}} - \frac{y_o}{s_y}$
21:                    if $0 < y_3 < 1$ then
22:                        $c_3 \leftarrow c\sqrt{s_x^2 + (s_y y_3 + y_o)^2} + f(1 - y_3) + g(s_2)$
23:                    end if
24:                end if
25:            end if
26:            if $c > b$ then and $s_x > 0$
27:                $x_4 \leftarrow 1 - \frac{s_y + y_o}{s_x}\frac{b}{\sqrt{c^2 - b^2}}$
28:                if $0 < x_4 < 1$ then
29:                    $c_4 \leftarrow b s_x x_4 + c\sqrt{s_x^2(1 - x_4)^2 + (s_y + y_o)^2} + g(s_2)$
30:                end if
31:            end if
32:            $vs, x, y \leftarrow \min\left((c_1, x_1, 0), (c_2, 0, 1), (c_3, 0, y_3), (c_4, x_4, 1)\right)$
33:        end if
34:    end if
35:    return $vs, x, y$
36: end function
```

both goal bracketing vertices, as in Figure 5.6b (Algorithm 11). The `IsGoal` function return true if a vertex is one of the four vertices that border a map cell with a goal point. The `GoalApproachParams` function returns the associated g_x, face, c, and b for a path that is approaching a map cell containing a goal and bordered by $s1$ and $s2$, where face is the north, south, east or west face of the cell. It also returns the offset of the goal point from the upper left corner of the cell, s_x and s_y. We use column-major linear indices to identify the vertices s. (See Section 5.5) The test on Line 9 uses this

Algorithm 10 Internal cost-to-goal values for nodes bracketing the goal state. The values c_x correspond to paths in Figure 5.6a.

```
function InternalGoalCost($s_{0,0}$, $s_{0,1}$, $s_{1,0}$, $s_{1,1}$, $s_x$, $s_y$, $c$, $b_N$, $b_S$, $b_E$, $b_W$)
    $g_N \leftarrow cs_x$
    $g_W \leftarrow cs_y$
    $g_S \leftarrow c(1 - s_x)$
    $g_E \leftarrow c(1 - s_y)$
    $c_1 \leftarrow$ InterpCostV2( $g_E$,  0,  $c$,  $b_E$,  $s_x$,      $1 - s_y$,  0,  True)
    $c_2 \leftarrow$ InterpCostV2( $g_N$,  0,  $c$,  $b_N$,  $1 - s_y$,  $s_x$,      0,  True)
    $c_3 \leftarrow$ InterpCostV2( $g_N$,  0,  $c$,  $b_N$,  $s_y$,      $s_x$,      0,  True)
    $c_4 \leftarrow$ InterpCostV2( $g_W$,  0,  $c$,  $b_W$,  $s_x$,      $s_y$,      0,  True)
    $c_5 \leftarrow$ InterpCostV2( $g_W$,  0,  $c$,  $b_W$,  $1 - s_x$,  $s_y$,      0,  True)
    $c_6 \leftarrow$ InterpCostV2( $g_S$,  0,  $c$,  $b_S$,  $s_y$,      $1 - s_x$,  0,  True)
    $c_7 \leftarrow$ InterpCostV2( $g_S$,  0,  $c$,  $b_S$,  $1 - s_y$,  $1 - s_x$,  0,  True)
    $c_8 \leftarrow$ InterpCostV2( $g_E$,  0,  $c$,  $b_E$,  $1 - s_x$,  $1 - s_y$,  0,  True)
    $g(s_{0,1}) \leftarrow \min(c_1, c_2)$
    $g(s_{0,0}) \leftarrow \min(c_3, c_3)$
    $g(s_{1,0}) \leftarrow \min(c_5, c_6)$
    $g(s_{1,1}) \leftarrow \min(c_7, c_8)$
    return $g(s_{0,0})$, $g(s_{1,0})$, $g(s_{1,1})$, $g(s_{0,1})$
end function
```

property to quickly determine the orientation of s_1 relative to s_2. If $s_x = s_y = 0.5$ then this test redundant.

5.2.4 Path Reconstruction

To begin path reconstruction, we compute the lowest cost-to-goal for a path starting at the non-integer start coordinate (Figure 5.7a, Algorithm 12). As before, there are eight possible configurations for the path, of which we choose the cheapest option. Unlike the cost-to-goal computation in Figure 5.6a, the values of g_N, g_S, g_E, g_W are interpolated using their bracketing field vertices, as in

$$g(s) = \min \begin{cases} bs_x + g(s_1) \\ cs_x + g(s_1) \\ \underbrace{b\left(1 - (s_y + y_o)\frac{b}{s_x\sqrt{c^2 - b^2}}\right)}_{x^*} + c\sqrt{(s_y + y_o)^2 + \left(\frac{b(s_y + y_o)}{\sqrt{c^2 - b^2}}\right)^2} + g(s_2) \\ c\sqrt{s_x^2 + \left(\frac{f}{s_y + y_o}\frac{1}{\sqrt{(s_y c)^2 - f^2}}\right)^2} + f\left(1 - \underbrace{\left(\frac{1}{s_y}\frac{f}{\sqrt{(s_y c)^2 - f^2}} - \frac{y_o}{s_y}\right)}_{y^*}\right) + g(s_2) \\ c\sqrt{1 + (s_y + y_o)^2} + g(s_2) \end{cases} \tag{5.82}$$

Algorithm 11 External cost-to-goal for cases where s_1 and s_2 are both goal bracketing vertices.

```
function BasicCostV2($s_1$, $s_2$, $g(s_1)$, $g(s_2)$, $c$, $b$)
    if IsGoal($s_1$) and IsGoal($s_2$) then
        $g(s_x)$, face, c, b, $s_x$, $s_y$ ← GoalApproachParams($s_1$, $s_2$)
        if face = South or face = North then        ▷ Edge on north or south face of goal cell
            $\delta \leftarrow s_y$
        else
            $\delta \leftarrow s_x$
        end if
        if $s_1 < s_2$ then
            $c_1$ ← InterpCostV2( $g(s_1)$, $g(s_x)$, $c$, $b$, 1,     $\delta$,          0, False)   ▷ Case 1
            $c_2$ ← InterpCostV2( $g(s_x)$, $g(s_2)$, $c$, $b$, 1, $1-\delta$,     $\delta$, False)   ▷ Case 2
        else
            $c_1$ ← InterpCostV2( $g(s_1)$, $g(s_x)$, $c$, $b$, 1, $1-\delta$,          0, False)   ▷ Case 1
            $c_2$ ← InterpCostV2( $g(s_x)$, $g(s_2)$, $c$, $b$, 1,     $\delta$, $1-\delta$, False)   ▷ Case 2
        end if
        return min($c_1$, $c_2$)
    else
        return BasicCostV1($g(s_1)$, $g(s_2)$, $c$, $b$)
    end if
end function
```

Algorithm 12 Line 2. These are estimates of the path cost to the goal *from* the point inside the map cell, not *to* the point inside the map cell. This means that the algorithm is not exactly symmetrical; a path from the start to the goal is not the same as a path from the goal to the start.

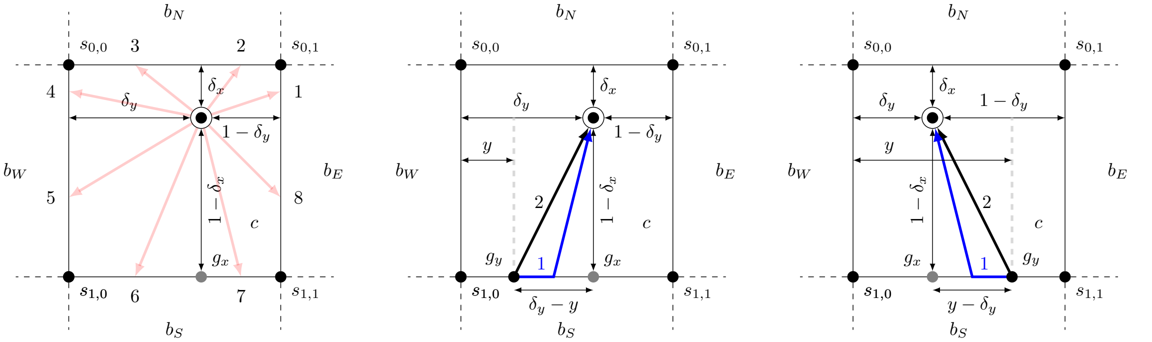
For consistency, each call to the interpolated cost function `InterpCostV2` in Algorithm 12 returns an x value. However, all these values should be 0 or 1 since the cost within the goal cell is uniform, and any non-direct path to the edge will not be optimal.

This function returns both a cost and a path waypoint coordinate in the form of two field vertices (s_1 and s_2) and an offset (y) on Line 22. The pair of vertices constitute a unit-vector in space with y as the offset. There are two ways to express a waypoint using this information. We standardize the form to be convenient for efficient translation to the discrete map coordinates used for storage. In our implementation, data is stored in a column-major format using unsigned integer indices. (See Section 5.5) We use these indices to to compactly identify $s1$ and $s2$. We standardize the coordinate by ensuring the linear index of s_1 is less than the linear index of s_2.

5.2.5 Path Reconstruction: Approaching Goal Cell

When computing the optimal path towards an edge that borders the goal point, we must employ a specialized algorithm. If our last step lands precisely on a vertex, then we proceed as in Algorithm 11. However, this algorithm does not return waypoints since it is designed for estimating the field values, and further, the more likely case is that we start on an interpolated waypoint. A more general algorithm is required.

There are 6 configurations to consider, illustrated in Figure 5.8. In these figures, the value g_x is



(a) Initializing reconstruction. (b) Completing reconstruction, case 1. (c) Completing reconstruction, case 2.

Figure 5.7: Path reconstruction for non-integer endpoints.

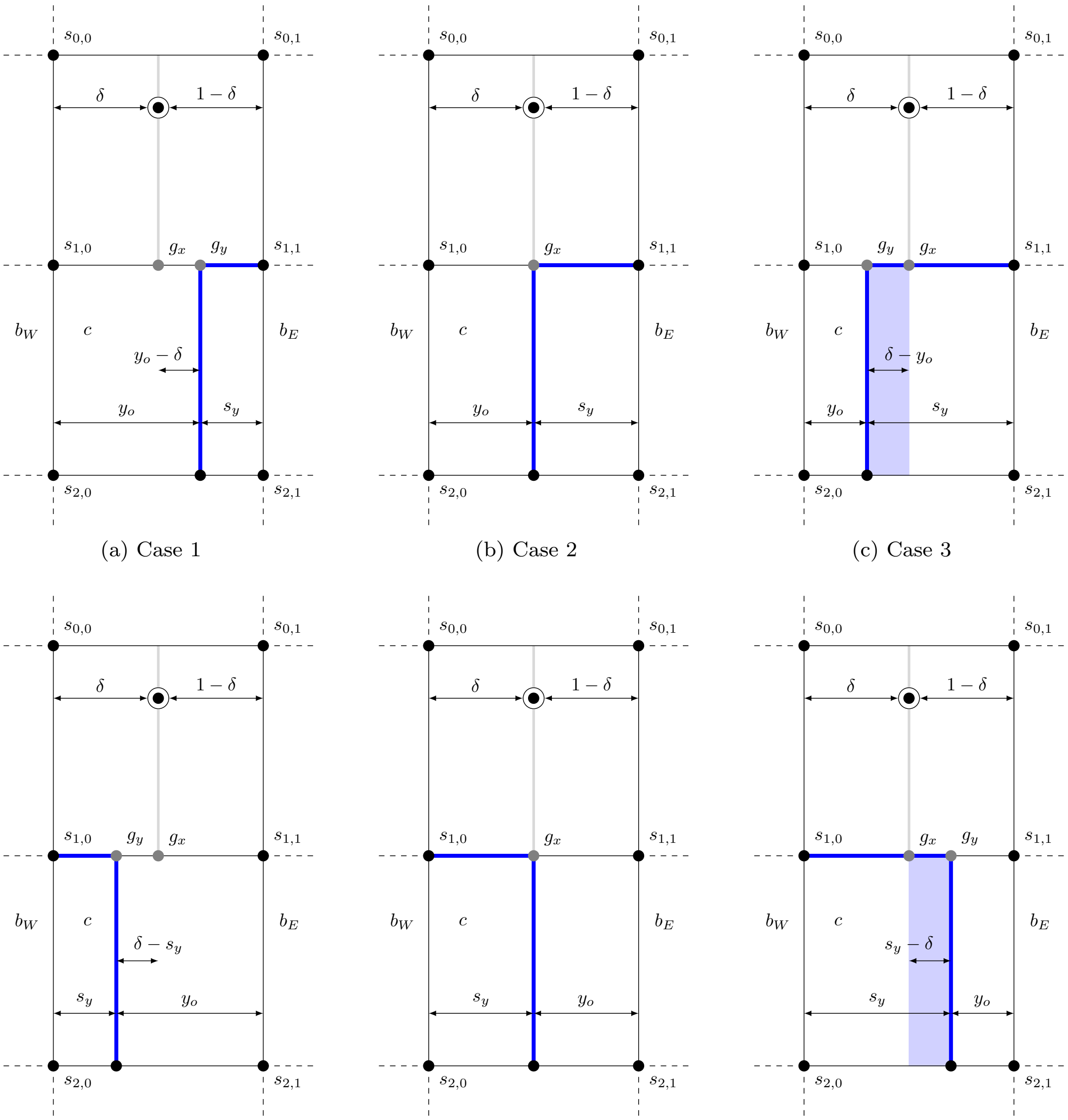
Algorithm 12 Internal cost-to-goal values for starting node for path reconstruction. The min operator selects the path coordinate tuple with the lowest cost.

```
 1: function StartCost($s_{0,0}$, $s_{0,1}$, $s_{1,0}$, $s_{1,1}$, $\delta_x$, $\delta_y$)
 2:     $g_N \leftarrow (g(s_{0,1}) - g(s_{0,0}))\,\delta_y + g(s_{0,0})$
 3:     $g_S \leftarrow (g(s_{1,1}) - g(s_{1,0}))\,\delta_y + g(s_{1,0})$
 4:     $g_W \leftarrow (g(s_{1,0}) - g(s_{0,0}))\,\delta_x + g(s_{0,0})$
 5:     $g_E \leftarrow (g(s_{1,1}) - g(s_{0,1}))\,\delta_x + g(s_{0,1})$
 6:     $c_1, x_1, y_1 \leftarrow$ InterpCostV2($g_E$, $g(s_{0,1})$, $C$, $C$, $1-\delta_y$, $\delta_x$, 0, False)      ▷ Case 1
 7:     $c_2, x_2, y_2 \leftarrow$ InterpCostV2($g_N$, $g(s_{0,1})$, $C$, $C$, $\delta_x$, $1-\delta_y$, 0, False)      ▷ Case 2
 8:     $c_3, x_3, y_3 \leftarrow$ InterpCostV2($g_N$, $g(s_{0,0})$, $C$, $C$, $\delta_x$, $\delta_y$, 0, False)      ▷ Case 3
 9:     $c_4, x_4, y_4 \leftarrow$ InterpCostV2($g_W$, $g(s_{0,0})$, $C$, $C$, $\delta_y$, $\delta_x$, 0, False)      ▷ Case 4
10:     $c_5, x_5, y_5 \leftarrow$ InterpCostV2($g_W$, $g(s_{1,0})$, $C$, $C$, $\delta_y$, $1-\delta_x$, 0, False)      ▷ Case 5
11:     $c_6, x_6, y_6 \leftarrow$ InterpCostV2($g_S$, $g(s_{1,0})$, $C$, $C$, $1-\delta_x$, $\delta_y$, 0, False)      ▷ Case 6
12:     $c_7, x_7, y_7 \leftarrow$ InterpCostV2($g_S$, $g(s_{1,1})$, $C$, $C$, $1-\delta_x$, $1-\delta_y$, 0, False)      ▷ Case 7
13:     $c_8, x_8, y_8 \leftarrow$ InterpCostV2($g_E$, $g(s_{1,1})$, $C$, $C$, $1-\delta_y$, $1-\delta_x$, 0, False)      ▷ Case 8
14:     $c, y, s_1, s_2 \leftarrow (c_1, \delta_x(1-y_1), s_{0,1}, s_{1,1})$
15:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_2, \delta_y + y_2(1-\delta_y), s_{0,0}, s_{0,1}))$
16:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_3, \delta_y(1-y_3), s_{0,0}, s_{0,1}))$
17:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_4, \delta_x(1-y_4), s_{0,0}, s_{1,0}))$
18:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_5, \delta_x + y_5(1-\delta_x), s_{0,0}, s_{1,0}))$
19:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_6, \delta_y(1-y_6), s_{1,0}, s_{1,1}))$
20:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_7, \delta_y + y_7(1-\delta_y), s_{1,0}, s_{1,1}))$
21:     $c, y, s_1, s_2 \leftarrow \min((c, y, s_1, s_2), (c_8, \delta_x + y_8(1-\delta_x), s_{0,1}, s_{1,1}))$
22:     return $c$,$y$,$s_1$,$s_2$
23: end function
```

pre-computed in Algorithm 10. That is, g_x is *not* linearly interpolated by the field values at s_{01} and s_{11}, which is a condition unique to the goal cell, as in Section 5.2.2.

Cases 1,2,4 and 5 do not have search intervals that coincide with the projection of the goal point onto the edge. The value of $g(s_1)$ is adjusted, but the cost computation is otherwise unchanged (Line 11).

Cases 3 and 6 are more complex. As with Figure 5.6b, there are two regions to consider. We



(a) Case 1

(b) Case 2

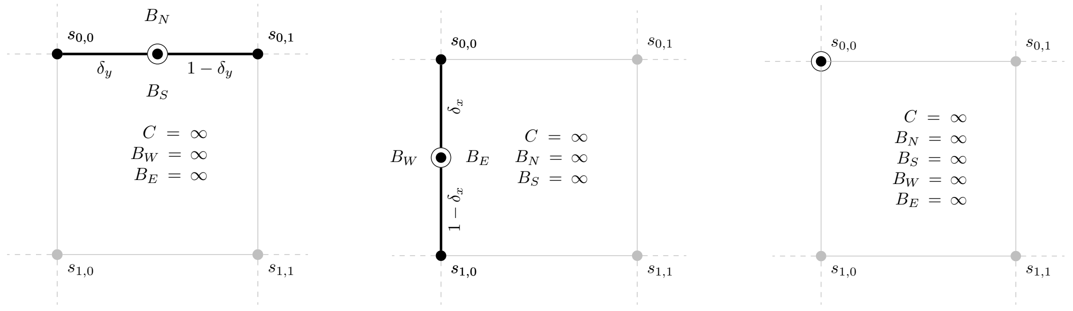
(c) Case 3

(d) Case 4

(e) Case 5

(f) Case 6

Figure 5.8: Approaching goal cell from linearly interpolated state. Cases 3 and 6 have two potential solutions.



(a) Case 1

(b) Case 2

(c) Case 3

Figure 5.9: Degenerate cases for internal cost-to-goal computation.

select the lowest cost solution among the two regions (Line 15), taking care to transform the solution into the same coordinate space as the initial problem (Line 19). Unlike Algorithm 12, the final transformation of the solution into standard coordinates is left to a higher level routine.

This algorithm is a more general version of Algorithm 11. To see this, set $s_x = s_y = 1$ and $y_o = 0$. If $s_1 < s_2$, then we reach case 3, with $\epsilon = \delta$ and $g_y = g(s_1)$. The parameters to the interpolation function are identical. The coordinate transform is not required for Algorithm 11. If $s_1 \geq s_2$, then $\epsilon = 1 - \delta$ and $g_y = g(s_1)$ and again, the parameters to the interpolation function are identical to Algorithm 11.

5.2.6 Path Reconstruction: Goal Edge to Goal State

The final part of the reconstruction is the path from the goal edge to the goal itself, as illustrated in Figures 5.7b and 5.7c. Figure 5.7b illustrates the case where we are at a non-integer coordinate on an edge bounding the goal state. This case is in addition to the normal path reconstruction cases that choose another edge waypoint. To understand why this is necessary, suppose that in Figure 5.7b, $C > 0$ and $B_N, B_W, B_E = 0$. It may very well make sense to travel around the perimeter of the goal state and reach the goal from the north. We select the path to the goal if it is cheaper than any path of equivalent cost. This is computed in Algorithm 14.

5.2.7 Degenerate Cases

It is important to check that the algorithms are well-behaved if the goal or start location is on a grid line or vertex (Figure 5.9).

Algorithm 9 is the core routine used by many of the special case computations. In particular if $s_x = 0$ or $s_y = 0$, cases C_1 and C_2 return solutions that make sense given the geometric constraint if one or both these conditions are true. However for case C_3, s_y must be greater than 0 since there is a division by s_y at line 20. Similarly for case C_4, a division by s_x at line 27 requires that $s_x > 0$. We have therefore added guards to protect these cases.

Algorithm 13 External cost-to-goal for cases where s_1 and s_2 are both goal bracketing states

```
function GoalApproachCost($s_1$, $s_2$, $g(s_1)$, $g(s_2)$, $c$, $b$, $s_x$, $s_y$, $y_o$)
    $g_x$, face ← GoalApproachParams($s_1$, $s_2$)
    if face = South or face = North then                ▷ Edge on north or south face of goal cell
        $\delta \leftarrow s_y$
    else
        $\delta \leftarrow s_x$
    end if
    if $s_1 < s_2$ then
        if $y_o \geq \delta$ then                                       ▷ Cases 1, 2
            $\epsilon \leftarrow y_o - \delta$
            $g_y \leftarrow (g(s_2) - g_x)\,\frac{\epsilon}{1-\delta} + g_x$
            $v$, $x$, $y$ ← InterpCostV2( $g_y$, $g(s_2)$, $c$, $b$, $s_x$, $s_y$, $y_o$, False)
        else                                                            ▷ Cases 3
            $\epsilon \leftarrow \delta - y_o$
            $g_y \leftarrow (g(s_1) - g_x)\,\frac{\epsilon}{\delta} + g_x$
            $v_1$, $x_1$, $y_1$ ← InterpCostV2( $g_y$, $g_x$, $c$, $b$, $s_x$, $\epsilon$, 0, False)
            $v_2$, $x_2$, $y_2$ ← InterpCostV2( $g_x$, $g(s_2)$, $c$, $b$, $s_x$, $s_y - \epsilon$, $\epsilon$, False)
            if $v_1 < v_2$ then
                $v \leftarrow v_1$   $x \leftarrow x_1$   $y \leftarrow \frac{1}{s_y}\epsilon y_1$
            else
                $v \leftarrow v_2$   $x \leftarrow x_2$   $y \leftarrow \frac{1}{s_y}\left((s_y - \epsilon)y_2 + \epsilon\right)$
            end if
        end if
    else
        if $\delta \geq s_y$ then                                       ▷ Cases 4, 5
            $\epsilon \leftarrow \delta - s_y$
            $g_y \leftarrow (g(s_2) - g_x)\,\frac{\epsilon}{\delta} + g_x$
            $v$, $x$, $y$ ← InterpCostV2( $g_y$, $g(s_2)$, $c$, $b$, $s_x$, $s_y$, $y_o$, False)
        else                                                            ▷ Case 6
            $\epsilon \leftarrow s_y - \delta$
            $g_y \leftarrow (g(s_1) - g_x)\,\frac{\epsilon}{1-\delta} + g_x$
            $v_1$, $x_1$, $y_1$ ← InterpCostV2( $g_y$, $g_x$, $c$, $b$, $s_x$, $\epsilon$, 0, False)
            $v_2$, $x_2$, $y_2$ ← InterpCostV2( $g_x$, $g(s_2)$, $c$, $b$, $s_x$, $s_y - \epsilon$, $\epsilon$, False)
            if $v_1 < v_2$ then
                $v \leftarrow v_1$   $x \leftarrow x_1$   $y \leftarrow \frac{1}{s_y}\epsilon y_1$
            else
                $v \leftarrow v_2$   $x \leftarrow x_2$   $y \leftarrow \frac{1}{s_y}\left((s_y - \epsilon)y_2 + \epsilon\right)$
            end if
        end if
    end if
    return $v$, $x$, $y$
end function
```

Appropriate corrections have been added to other algorithms to protect against these edge cases. Finally, we restrict the reconstructed path from visiting a node that has already been visited to prevent cycles. The next lowest cost path forward is selected when this case is encountered.

Algorithm 14 Path reconstruction for points proximal to the goal.

```
function GoalTerminalCost($s_1$, $s_2$, $y$)
    $g_x$, face $\leftarrow$ GoalApproachParams($s_1$, $s_2$)
    if face = North then
        $B \leftarrow B_N$    $g_x \leftarrow g_N$    $s_x \leftarrow |y - \delta_y|$    $s_y \leftarrow \delta_x$
    end if
    if face = South then
        $B \leftarrow B_S$    $g_x \leftarrow g_S$    $s_x \leftarrow |y - \delta_y|$    $s_y \leftarrow 1 - \delta_x$
    end if
    if face = East then
        $B \leftarrow B_E$    $g_x \leftarrow g_E$    $s_x \leftarrow |y - \delta_x|$    $s_y \leftarrow 1 - \delta_y$
    end if
    if face = West then
        $B \leftarrow B_W$    $g_x \leftarrow g_W$    $s_x \leftarrow |y - \delta_x|$    $s_y \leftarrow \delta_y$
    end if
    return InterpCostV2( $g_x$,    0,    $C$,    $B$,    $s_x$,    $s_y$,    0,    True)
end function
```

5.2.8 Max Margin Planning Loss with Field D*

Table 5.2: Incremental path cost and map gradient for general case.

General Cost Computation

Case	$\Delta g(s)$	$\frac{\partial \Delta g(s)}{\partial b}$	$\frac{\partial \Delta g(s)}{\partial c}$
$x = 1,\ y = 0$	bs_x	s_x	0
$x = 0,\ y = 0$	cs_x	0	s_x
$x = 0,\ y = 1$	$c\sqrt{s_x^2 + s_y^2}$	0	$\sqrt{s_x^2 + s_y^2}$
$x > 0,\ y = 1$	$bs_x x + c\sqrt{(s_x(1-x))^2 + s_y^2}$	$s_x x$	$\sqrt{(s_x(1-x))^2 + s_y^2}$
$x = 0,\ y > 0$	$c\sqrt{s_x^2 + (s_y y)^2}$	0	$\sqrt{s_x^2 + (s_y y)^2}$

The path cost of a Field D* solution is the sum of the cost of individual path segments. Tables 5.2 and 5.3 shows the incremental cost of each path segment. It is immediately apparent that the total cost is expressible as the dot product of a vectorized cost map and a generalized state visitation vector. As with A*, the map gradient is sparse. Unlike A*, the cost of a path segment may depend on more than one map cell. The visitation vector is recovered by Algorithm 15.

Table 5.3: Incremental path cost and map gradient for external cost-to-goal.

External Cost-to-Goal Computation

Case	$\Delta g(s)$	$\frac{\partial \Delta g(s)}{\partial b}$	$\frac{\partial \Delta g(s)}{\partial c}$
$x = 1,\ y = 0$	b	1	0
$x = 0,\ y = 0$	c	0	1
$x = 0,\ y = 1$	$c\sqrt{1 + (s_y + y_o)^2}$	0	$\sqrt{1 + (s_y + y_o)^2}$
$x > 0,\ y = 1$	$bx + c\sqrt{(1-x)^2 + (s_y + y_o)^2}$	x	$\sqrt{(1-x)^2 + (s_y + y_o)^2}$
$x = 0,\ y > 0$	$c\sqrt{1 + (s_y y + y_o)^2}$	0	$\sqrt{1 + (s_y y + y_o)^2}$

Algorithm 15 Path visitation for Field D* paths. P is a sequence of path coordinates. M_r and M_c are the number of rows and columns in the map, M. $P_r(i)$ and $P_c(i)$ access the row and column of the i-th path element.

```
function PathVisitationFDStar($P$, $M$, $M_r$, $M_c$)
    V $\leftarrow \mathbf{0}$                                          ▷ Zero vector $\in \mathbb{R}^{M_r M_c}$
    for $i$ in $0...|P| - 2$ do
        $r_0 \leftarrow P_r(i) + 0.5$        $c_0 \leftarrow P_c(i) + 0.5$         ▷ Field coordinates to map coordinates
        $r_1 \leftarrow P_r(i+1) + 0.5$      $c_1 \leftarrow P_c(i+1) + 0.5$
        $r \leftarrow$Int$(\min(r_0, r_1))$    $c \leftarrow$Int$(\min(c_0, c_1))$     ▷ Integer map coordinates for cell $C$
        $v \leftarrow \sqrt{(c_0 - c_1)^2 + (r_0 - r_1)^2}$
        $B \leftarrow \infty$
        $C \leftarrow \infty$
        if $(r < M_r)$ and $(c < M_c)$ then
            $C \leftarrow M(r, c)$
        end if
        if $r_0 = r_1$ and $r_o <$FLOOR$(r_0)+\epsilon$ then          ▷ Horizontal boundary between cells
            if $r > 0$ then
                $B \leftarrow M(r-1, c)$                               ▷ $B$ is cell above $C$
            end if
            if $B < C$ then
                $r \leftarrow r - 1$                                   ▷ $B$ contributes to cost and not $C$
            end if
        end if
        if $c_0 = c_1$ and $c_o <$FLOOR$(c_0)+\epsilon$ then          ▷ Vertical boundary between cells
            if $c > 0$ then
                $B \leftarrow M(r, c-1)$                               ▷ $B$ is cell left of $C$
            end if
            if $B < C$ then
                $c \leftarrow c - 1$                                   ▷ $B$ contributes to cost and not $C$
            end if
        end if
                                                                       ▷ Other cases internal to cell $C$
        $V(r + M_r c) \leftarrow V(r + M_r c) + v$                     ▷ Accumulate visitation count
    end for
    return $V$
end function
```

5.3 Accelerated Learning

We performed an analysis of our algorithm to determine the performance-limiting factors. We use PyTorch ([102]) as our deep learning framework. PyTorch is a mix of CPU and GPU-optimized deep learning routines coordinated by Python scripts, which combines the performance of customized, low-level code with the flexibility of Python. The result of our analysis highlights the need for an accelerated loss computation, the benefits of which are most clearly illustrated in hindsight by the figures in Table 5.4.

In Table 5.4, we see that the GPU accelerated code is as much as nine times faster than the CPU version of the code, given the same number of CPU cores (light blue). At least 6 CPU cores are required to achieve peak performance with the GPU. These cores are primarily used to transfer data from the disk to the GPU. The light gray row shows that most of the speed improvement comes from the CUDA implementation of Field D* and not the CUDA version of the MMP loss. Note that these metrics utilize only one GPU, and a three GPU system would be data-starved, given an 18-core CPU.

We reduce the number of available cores to emulate typical server configurations (as of 2021). For example, an Amazon EC2 P2 instance has a CPU core/GPU ratio of 4:1. A more advanced P4 instance has a CPU core/GPU ratio of 12:1, but these high-performance machines are generally too expensive for exploratory research.

The CPU loss algorithm tested in Table 5.4 is already highly optimized. Our research began with a 4-core desktop CPU, with the loss algorithm at least 35 times slower than the GPU accelerated version. The improvement from the original un-optimized version of the CPU loss function (not shown) is even more considerable. Early training runs took days to complete but now finish in a few hours. Figure 5.10 illustrates how a hybrid CPU/GPU system operates during training. In all cases, the CPU is responsible for loading training data and sending it to the GPU for the forward pass of the network. This is not shown and is usually an asynchronous background process that prepares and sends the next batch of data ahead of the next training step.

The loss function Figure 5.10a receives the result of the forward pass of the network. The CPU computes the loss and sometime later returns the gradients for the backward pass through the network. In our case, these are several hundred cost maps and associated gradients amounting to a few megabytes of data. While the amount of data is not large, this introduces communications latency and forces the GPU to idle while the CPU is busy.

There are methods for distributed deep learning that allow for looser coupling between computing nodes. One could imagine a pipeline-parallel system such as in Figure 5.10b. Kosson *et al.* [115] is an example of a fine-grained form of pipelined backpropagation. However, this is only helpful if the computational time for the CPU stage is approximately the same as for the GPU stages. This is not the case for our task when considering commercial deep learning cloud instances that do not provide a large number of CPU cores. The latency of the pipeline stages will remain unbalanced, which will limit any performance gain according to Amdahl's law.

Translating an algorithm for use on a GPU is not yet an automated process. The GPUs

Table 5.4: Analysis for accelerated loss computation. Test map tensor is 128 × 128, N=200. CPU is Intel i9-10980XE, 18 Cores at 3.0 GHz, 2 × NVIDIA RTX 2080Ti. One GPU used in this benchmark.

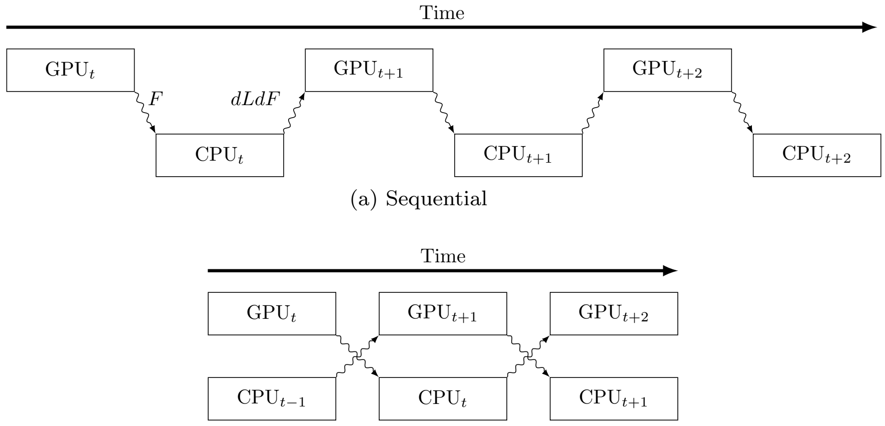
CUDA	CPU Cores	Time (sec)	Iter/sec	Speedup(GPU)	Speedup (4-core)
Y	18	15.82	63.20	8.92	35.28
Y[a]		125.65	7.96	1.12	4.44
N		141.16	7.08	1.00	3.85
Y	8	15.86	63.05	18.31	35.19
N		290.46	3.44	1.00	1.92
Y	6	15.79	63.36	23.60	35.35
N		372.58	2.68	1.00	1.50
Y	4	62.91	15.90	8.87	8.87
N		558.17	1.79	1.00	1.00

[a] CPU Field D* with GPU loss computation.

used in this research are programmed with C++, yet despite using the same language as our CPU implementation, the algorithms are quite different. This is primarily because the Single Instruction Multiple Thread (SIMT) programming model of the GPU belies a vastly different hardware architecture, with the potential for tremendous parallelism but only after careful attention to code branching and memory access patterns. These details are not as important for CPUs, which devote large amounts of hardware towards hiding the latency introduced by branching code and non-ideal memory access patterns. The GPU sacrifices this for more computational hardware.

A very high-end server in 2021 may have 64 cores (Ryzen Threadripper 3990x), each independent of the other. In contrast, a GPU typically contains thousands of computational cores, giving a theoretical peak floating-point performance far in excess of any CPU. This is possible because the GPU does not have the physical hardware that a CPU uses to excel in generic tasks. Instead, it specializes in tasks where computation is tightly coupled to a regular computational graph that admits highly parallel arithmetic and memory operations. While frameworks such as OpenCL [116] strive to automate the process of transforming plain C++ into accelerated code for the GPU, our experiments with this have not been productive.

We will next show how the Field D* algorithm and the Fréchet metric are implemented on the GPU to accelerate overall system performance by removing the CPU computational bottleneck. We omit the conversion of the Max Margin Planning loss and the Modified Hausdorff Metric because they are of less technical interest. The MMP loss is not computationally expensive and does not require extensive modification to translate to the GPU. The Hausdorff metric is implemented as a simplified version of the accelerated Fréchet metric.



(a) Sequential

(b) Parallel

Figure 5.10: Different modes for parallel computation in a heterogeneous system.

5.3.1 Accelerated Field D*

The original Field D* algorithm for the interpolated cost computation is not well-suited for GPU applications. In the following text, we will discuss the computational bottlenecks that prevent effective utilization of GPU resources and how we modify the algorithm to eliminate them.

The GPU is central to many machine learning applications because it has exceedingly high memory and computational throughput. Like most work in deep learning, our algorithm makes extensive use of this capability, albeit invisibly through PyTorch. However, unlike most applications, our loss function contains a planning step that is not easily expressed with the computational primitives integral to PyTorch or other CUDA libraries.

An NVIDIA GPU partitions processing cores into groups of 32 threads, called warps, which are scheduled as a unit and execute the same instructions in lock-step[4]. If some threads enter a code branch, all other threads will wait until the branch completes and the warp reaches a common point in the code. If one thread must execute a branch (for a condition that occurs on a map border, for example), the remaining 32 threads must wait. A CPU is less affected by this problem because each core is usually fully independent, runs multiple tasks to hide latency, and devotes significant resources to branch prediction and memory caching.

5.3.2 Bellman-Ford-Moore Field Computation

Much like Dijkstra's Algorithm, Field D* updates the cost field by expanding a single frontier node at each time step. The location of that node is hard to predict, which also makes it hard to hide memory access latency by pre-loading data. The algorithm is computationally efficient, but it is also a poor fit for GPU architectures.

The Bellman-Ford algorithm (Bellman [114]), also called the Bellman-Ford-Moore algorithm, is

[4] There are exceptions to this rule depending on the hardware and the instruction being executed.

an alternative to Dijkstra's algorithm and its derivatives. This algorithm can fully utilize the GPU multiprocessors and predictably access memory by assigning one thread to each row of the map. While in comparison to Dijkstra's algorithm, it requires far more computational steps to complete, these operations occur on a massively parallel architecture and require little coordination between threads.

We observe that the field update in Equation (5.82) are additive with respect to $g(s_1)$ and $g(s_2)$, the field values. Therefore, the field values are monotonically decreasing as the algorithm progresses and the inductive proof of convergence for Bellman-Ford is still correct. Ferguson *et al.* [105] must have followed similar reasoning when extending A* to use the field update equations.

The runtime of Bellman-Ford is $\mathcal{O}(mn)$. While not work efficient, it fully utilizes the far more numerous processors on the GPU, enhancing the speed of the overall system at the cost of redundant computations. The Bellman-Ford algorithm is intimately related to value iteration and has similar runtime. While our initial intent in selecting Max Margin Planning was to utilize the CPU for efficient planning with A*/Field D*, the move to Bellman-Ford on the GPU has made the difference less relevant.

There is extensive research on accelerating the fundamental algorithm of computing the shortest path through a graph. Much of it focuses on very large generic graphs, as opposed to the regular, locally connected graph discussed here. This research generally devotes great effort to increasing work efficiency by avoiding redundant relaxations. Instead, we choose to minimize main memory access, code branching, and code complexity so that the worker threads and the floating-point units are fully utilized.

The Δ-stepping algorithm (Meyer *et al.* [117], [118]) and variants (Duriakova *et al.* [119] and Chakaravarthy *et al.* [120]) are hybrids between Dijkstra's algorithm and the Bellman-Ford algorithm that use parallelism intelligently to maintain high work efficiency. When the parameter $\Delta = 1$, they are a version of Dijkstra's algorithm. When $\Delta = \infty$, they behave like Bellman-Ford. Radius stepping is a related algorithm (Blelloch *et al.* [121]). These algorithms are more suitable for very large problems solved with CPU clusters that have non-uniform memory access (NUMA) constraints. Generally, parallelism comes from breaking the graph into smaller pieces that local processors solve. These graphs are irregular, and the algorithms are highly branching. Peng *et al.* [122], Prasad *et al.* [123], and Nazarifard *et al.* [124] are different approaches that are also better suited for irregular graphs on multi-core systems.

Busato *et al.* [125], Surve *et al.* [126], Davidson *et al.* [127], Kumar *et al.* [128], and Harish *et al.* [129] are various GPU implementations to solve the shortest path problem on general graphs that are defined with adjacency lists. Because our topology is locally connected and regular, we can partition the work more efficiently. In particular, modern GPUs have features that were not available to older work, such as Harish *et al.* [129].

5.3.3 Divergent Branches in Field Computation

We have modified Algorithm 6 to eliminate branching in the field computation[5]. Recall that all threads in a 32-thread group must execute the same code at the same time. In the case of a branch, all other threads wait until the branching thread(s) complete their branch. This means that parallelism may be reduced by a factor of 32.

Algorithm 16 Accelerated cost computation. Excess branches were removed to eliminate pipeline stalls.

```
1: function BasicCostGPU($g(s_1)$, $g(s_2)$, $c$, $b$)
2:     $v_1 \leftarrow \min(b, c) + g(s_1)$
3:     $f \leftarrow g(s_1) - g(s_2)$
4:     $z \leftarrow \min(f, b)$
5:     $a \leftarrow c/z$
6:     $y \leftarrow \min(1.0, \frac{1.0}{\sqrt{a^2 - 1.0}})$
7:     $v_2 \leftarrow (c\sqrt{y^2 + 1} + z(1 - y) + g(s_2)$
8:     return $\min(v_1, v_2)$
9: end function
```

We have applied the following optimizations to Algorithm 6 to arrive at Algorithm 16.

- The optimal x and y are not used when computing $g(s)$. (Deleted)
- If $b = \infty$ or $c = \infty$, the result will be ∞. No need for a special test for this condition. Returning early actually increases runtime since all other threads in the warp will be delayed.
- Cases 4 and 5 in Algorithm 6 are merged by observing the symmetry in Equation (5.44), swapping f and b.
- A min operator on Line 6 ensures that the solution is not greater then 1.0. When $y = 1.0$, the solution is the same as Case 3 in Algorithm 6. If the radical on line 6 returns a NaN (when $a < 1$), IEEE 754 specifies that min(1.0f, NaN) = 1.0f.

These modifications transform control-flow dependencies into data-flow dependencies. The floating-point unit of the GPU has tremendous throughput if the instruction pipeline is full, which is often difficult to achieve. This optimization is also beneficial to our CPU implementation.

5.3.4 Efficient Memory Access

The NVIDIA GPU architecture generally does not have a complex system for caching memory accesses. The GPU is intended for massively parallel computation, and memory access latency is instead hidden with context swapping between warps to ensure that the execution units are constantly occupied. Each GPU multiprocessor does have a local shared memory with lower access latency than the main memory. This memory is about 49KB For the GPU (NVIDIA RTX 2080 Ti), a fraction of the 11 GB of main memory. Both the main memory and shared memory have

[5] The interpolated cost computation used for path reconstruction is not similarly optimized since it accounts for a fraction of the runtime.

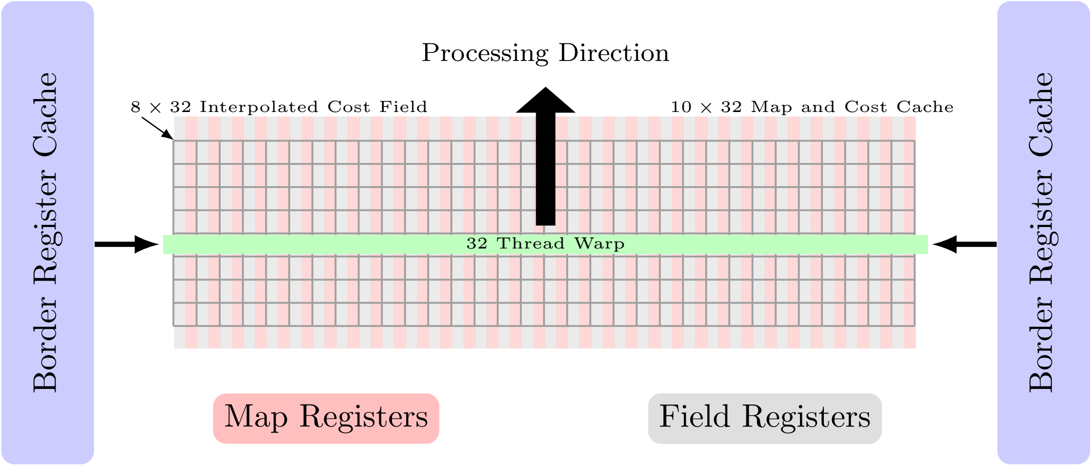


Figure 5.11: A single CUDA warp iterates until convergence over a patch of the Field D* cost map, carefully keeping all data in local registers to minimize access latency and maximize floating-point utilization.

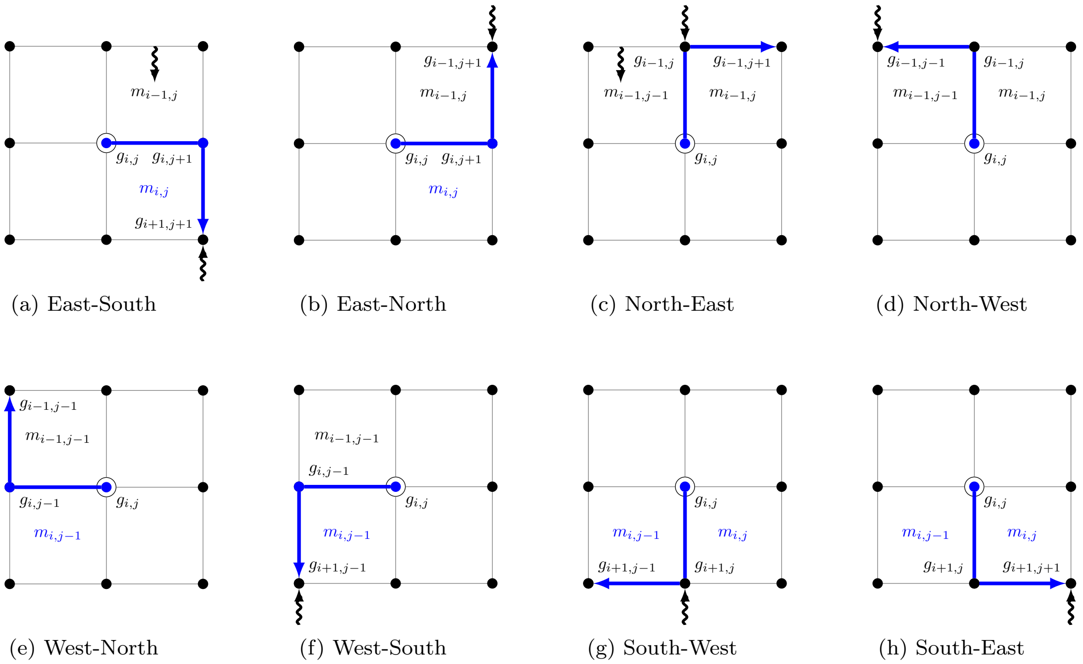
restrictions to ensure efficient access. In general, each thread in the warp must access memory as part of a single 1024-bit contiguous read. Any other access pattern leads to delay. For this reason, we use shared memory mostly for communication between threads warps.

Instead, we efficiently use register memory. Each multiprocessor has 256KB of register memory, more than five times the available shared memory. Registers are the fastest memory with no access restrictions, but a thread's registers are not directly visible to other threads and can not be shared with other warps. The CUDA warp shuffle operation allows threads in a warp to communicate by exchanging register contents directly.

We partition the cost field into $32 \times N$ non-overlapping blocks of work, large enough to utilize all available register memory. Within a block, each thread stores one row of the map and one row of the cost field in local registers (Figure 5.11). The rows are length $N + 2$, but the block updates only the central N field values. The two excess values are constant and required to compute the central values (Figure 5.4b).

Algorithm 17 is a version of Bellman-Ford that iterates over the block until no cell has changed by more than $\delta = 10^{-5}$, as broadcast by a warp reduction. The computation proceeds similarly to Algorithm 7, substituting memory accesses with warp shuffle operations for variables not in the thread's register cache, depicted by wavy arrows in Figure 5.12. An arrow pointed downwards transfers memory from the thread above. An arrow pointed upwards transfers a value from the thread below. All shuffle operations between neighboring threads in a warp execute concurrently. All threads execute the East-South operation, then the East-North operation, etc., in lockstep.

The order of the field updates is different from Algorithm 7. This order was selected to carefully avoid redundant transfers between threads and to quickly use transferred values so as not to occupy registers unnecessarily. If the thread is the first or last thread in the warp, then there are no neighboring threads for some of these transfers. In this case, a second shuffle operation reads from a



(a) East-South

(b) East-North

(c) North-East

(d) North-West

(e) West-North

(f) West-South

(g) South-West

(h) South-East

Figure 5.12: Sequence of operations for a single worker thread in Figure 5.11. Blue values are stored locally in a register. Values in black are transferred using a warp shuffle (wavy line) from a neighboring thread. The top-most and bottom-most threads in the warp with no neighbors use a warp shuffle to read from a register cache distributed among all threads in the warp.

Algorithm 17 Relaxation process for single thread in warp of 32. The $\hat{g}(\cdot)$ and $\hat{m}(\cdot)$ return register cache values of g and m as described in Figure 5.12.

```
function FieldDStarBlock($g$,$m$,$row_0$,$col_0$)
    parfor $i \in [row_0 \,..\, row_0 + 31]$ do
        $\delta_{warp} = \infty$
        while $\delta_{warp} > 10^{-5}$ do
            $\delta_{row} = 0$
            for $j \leftarrow col_0 \,..\, col_0 + N - 1$ do
                $v_8 \leftarrow$ BasicCostGPU($\hat{g}(s_{i,j+1})$, $\hat{g}(s_{i+1,j+1})$, $\hat{m}(s_{i,j})$, $\hat{m}(s_{i-1,j})$ )      ▷ ES
                $v_1 \leftarrow$ BasicCostGPU($\hat{g}(s_{i,j+1})$, $\hat{g}(s_{i-1,j+1})$, $\hat{m}(s_{i-1,j})$, $\hat{m}(s_{i,j})$ )      ▷ EN
                $v_2 \leftarrow$ BasicCostGPU($\hat{g}(s_{i-1,j})$, $\hat{g}(s_{i-1,j+1})$, $\hat{m}(s_{i-1,j})$, $\hat{m}(s_{i-1,j-1})$ )  ▷ NE
                $v_3 \leftarrow$ BasicCostGPU($\hat{g}(s_{i-1,j})$, $\hat{g}(s_{i-1,j-1})$, $\hat{m}(s_{i-1,j-1})$, $\hat{m}(s_{i-1,j})$ )  ▷ NW
                $v_4 \leftarrow$ BasicCostGPU($\hat{g}(s_{i,j-1})$, $\hat{g}(s_{i-1,j-1})$, $\hat{m}(s_{i-1,j-1})$, $\hat{m}(s_{i,j-1})$ )  ▷ WN
                $v_5 \leftarrow$ BasicCostGPU($\hat{g}(s_{i,j-1})$, $\hat{g}(s_{i+1,j-1})$, $\hat{m}(s_{i,j-1})$, $\hat{m}(s_{i-1,j-1})$ )  ▷ WS
                $v_6 \leftarrow$ BasicCostGPU($\hat{g}(s_{i+1,j})$, $\hat{g}(s_{i+1,j-1})$, $\hat{m}(s_{i,j-1})$, $\hat{m}(s_{i,j})$ )      ▷ SW
                $v_7 \leftarrow$ BasicCostGPU($\hat{g}(s_{i+1,j})$, $\hat{g}(s_{i+1,j+1})$, $\hat{m}(s_{i,j})$, $\hat{m}(s_{i,j-1})$ )      ▷ SE
                $v = \min(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_7)$
                $\delta_{row} = \max(\delta_{row}, |\hat{g}(s_{i,j}) - v|)$
                $\hat{g}(s_{i,j}) = v$
            end for
            $\delta_{warp}$ = WarpReduceMax($\delta_{row}$)      ▷ Max reduction across all threads in warp
        end while
    end parfor
    return $\delta_{warp}$
end function
```

border cache (Figure 5.11) that is distributed across all threads.

The register cache is pre-loaded at the invocation of Algorithm 17. Out-of-bounds accesses are padded with infinity. At the conclusion of Algorithm 17, the updated values for $\hat{g}$ are written back to the main memory as a sequence of aligned coalesced writes.

At a higher level, Algorithm 18 coordinates a team of warps that relax non-overlapping map blocks until all blocks report a maximal change less than $\delta = 10^{-5}$ for any cell between iterations.

5.3.5 Accelerated Fréchet Metric

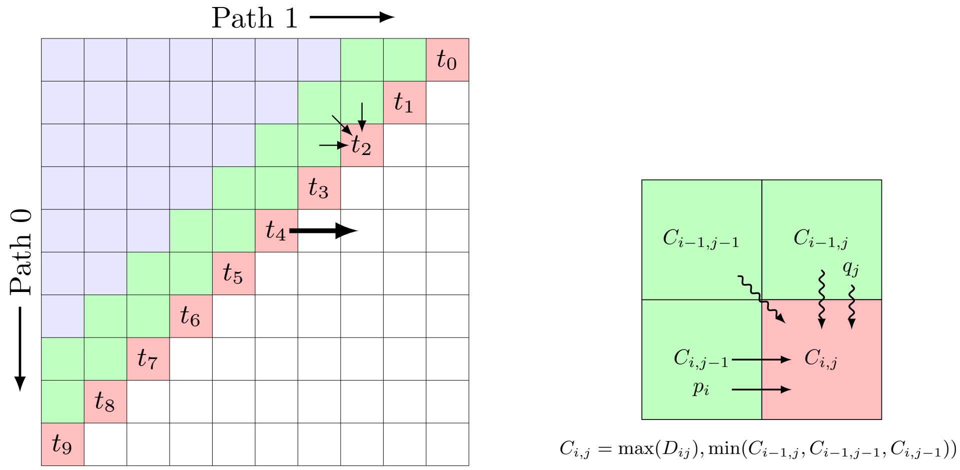
The solution table for the accelerated Fréchet metric can be computed in parallel on an expanding wavefront, with each thread executing the equations outlined in Section 3.4.2.

Figure 5.13a illustrates the motion of a collection of threads on this wavefront. Red indicates the active area, and blue indicates completed computations, which are discarded. Only results in the green region influence the active region. Figure 5.13b illustrates the local computation for each thread. Wavy arrows show intra-warp communication between neighboring threads. Straight arrows represent transfers from register memory. Only the lead thread, t_0, accesses shared memory to retrieve path points for the next computation step, making the computation extremely efficient.

We tested the performance of our algorithm and report the results in Table 5.5. This test first

Algorithm 18 Bellman-Ford-Moore variant of Field D*

```
function BellmanFordFieldDStar
    $\delta \leftarrow \infty$
    parfor $s \in States$ do
        $g(s) \leftarrow \infty$
    end parfor
    while $\delta > 10^{-5}$ do
        parfor $block$ in $Blocks$ do
            $row_0, col_0 \leftarrow$ Origin($block$)
            $\delta_{block}$ = FieldDStarBlock($g$, $m$, $row_0$, $col_0$)
            $\delta$ = BlockReduceMax($\delta_{block}$)
        end parfor
    end while
end function
```



(a) Wavefront expansion

(b) Local computation

Figure 5.13: Fréchet coupling metric computed with expanding wavefront. Values in red are active. Values in green are cached. Values in blue are past results that are discarded.

uses Field D* to compute a path for a map and then compares it to the ground truth using the Fréchet metric. The map size is 128×128 and we set $\delta_{dF} = 0.1$ pixels.

The performance of the GPU algorithm is about five times faster than the CPU version for $N = 128$ and increases to almost ten times faster for $N = 384$. This advantage will be even more dramatic in comparison to CPUs with fewer cores or with larger map sizes and longer paths.

5.4 Summary

Our objective, to imitate human-generated trajectories given a command and image, is only possible with a path planner capable of expressing fluid paths. The Field D* algorithm has this property, and it uses a similar efficient dynamic program as A*. The Field D* path cost is conveniently

Table 5.5: Runtime in seconds for various versions of Fréchet metrics computation over 1000 iterations. Intel i9-10980XE, 18 Cores at 3.0 GHz, 2×NVIDIA RTX 2080Ti.

N	128	256	384
CPU	45.6	83.6	164.7
GPU	9.2	12.5	16.6

expressed as the dot product of the vectorized cost map and a generalized state visitation count, making it ideal for extending Max Margin Planning. This is a unique contribution, to the best of our knowledge.

We have presented a detailed description of our Field D* implementation and show that it translates well to massively parallel architectures after appropriate simplification and optimization. While the availability of high-performance, highly parallel computational engines in the form of GPUs has catalyzed the field of deep learning, no accelerated planning libraries were available at the commencement of this work.

The initial planner used an 18-core high-performace CPU to minimize GPU idle time and maximize throughput. These CPUs are not common in commercial deep learning hardware, such as the Amazon Cloud, which uses smaller CPU instances as a conduit for moving data to the GPU. The disparity grows more significant with powerful GPUs in clusters. We eliminate this problem by moving the entire planning and loss computation to the GPU. While time-consuming, this work was necessary and is a valuable contribution to others researching deep learning and planning algorithms.

5.5 Appendix

Our implementation of Field D* organizes map memory in a column-major format. Each vertex has a linear integer index. This makes memory addressing trivial and compactly stores the 2D coordinate as a single integer, saving CPU cache space. With this scheme, it is easy to compute the indices of neighbors without requiring any comparisons for coordinate overflow or underflow, except for the border regions. We resolve this by carefully padding the map with regions initialized to infinity.

Cells in the border region are never updated by Field D* because they have infinite cost and are never returned by the heap for expansion. The CPU does not need to test for map boundary conditions, which increases execution speed by reducing the branching of the algorithm considerably.

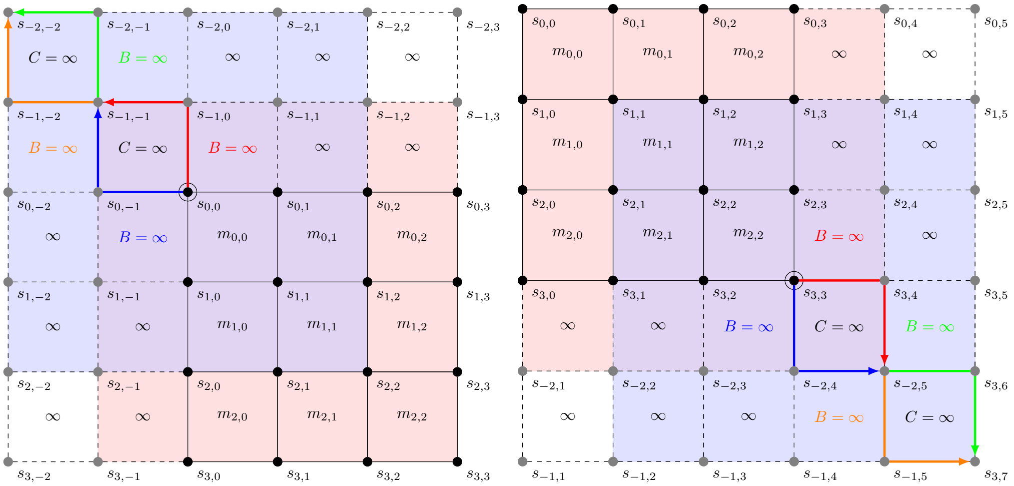
5.5.1 Memory Layout

Figure 5.14 demonstrate the extreme cases for the map. Given and $N \times M$ map, we have $(N+1) \times (M+1)$ states enveloping the map. In Figure 5.14a, we are performing an update centered on $s_{1,1}$. This involves updating the g values of the neighbors of $s_{1,1}$, which includes $s_{0,0}$, denoted with a circle. An update of $s_{0,0}$ will require the g values of all the nodes in the blue rectangle and associated map values. The red and blue paths denote the two most extreme cases, each requiring the map value C and the B with the matching color.

One solution to this boundary problem is to have a guard condition that prevents out-of-bounds memory access. However, this condition will need to execute for every memory access. An alternative is to pad the map with values so that specific nodes are never expanded but can always be accessed.

In the example of the red path, since C and B are both infinities, the cost of any path will also be infinity. However, the algorithm in Ferguson *et al.* [105] computes the minimum cost expansion for the neighbor-of-the-neighbor of a node. Therefore, when computing the field value for $s_{0,0}$, the algorithm may access $s_{-2,-2}$.

Two additional rows and columns are required at the top and left sides of the map. The bottom and right sides of the map must also be padded. However, because the map is a linear array of memory, rows 4 and 5 will alias to cells at the top of the map, one column to the right, which is already padded.



(a) Upper left

(b) Lower right

Figure 5.14: Extreme update cases for padded map. Red region contains map and field values that are valid. Blue region are values that field computation will access.

Figure 5.15 shows the total memory padding for the map. Two other cells are required in the upper right since accesses from the lower right of the map are aliased here. All map and g values in the red regions are initialized to infinity.

The total memory allocated is $(M + 3) \times (N + 5) + 2$. The column stride is $M + 3$ and regular map pixels begin at $(2, 2)$ which is a linear address of $2(M+3)+2$. In general, map pixels correspond to padded pixels with the formula:

$$l = (col + 2)(M + 3) + (row + 2)$$

These strategies simplify the code by removing almost all boundary and coordinate overflow/underflow

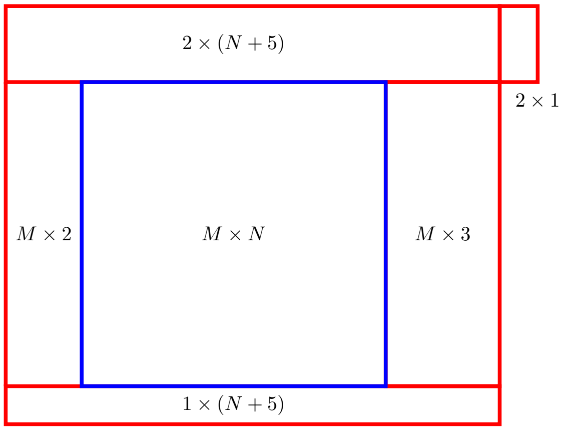


Figure 5.15: Memory padding for map and g. Blue region contains original map. Red region padded with infinity.

checks.

Chapter 6

Datasets & Results

In this chapter, we present two datasets that we use to evaluate the performance of our system. Annotating images is ordinarily expensive and time-consuming. Without existing datasets and a proven system, we have decided to create a synthetic dataset to trial our design. We have also developed a semi-synthetic dataset, which uses real aerial images that have been annotated with major features, which we augment with synthesized commands and trajectories. The step towards producing our own datasets was taken only after investigating existing work that might be tangentially suitable for training a deep network for robot navigation. By first working with synthetic data, we can refine our techniques before investing the time to develop realistic, human-annotated datasets.

We present an error analysis of the algorithm when trained on synthetic data in Section 6.2 so that we can understand what kinds of scenarios the model has difficulty learning. This analysis will be more useful in the future when the dataset is expanded to include more elaborate commands.

In Section 6.3, we perform a clustering analysis of the FiLM modulation vectors to lend insight into how the algorithm transforms image features. We examine how language affects the cost map by directly comparing cost maps produced by the deep network, with and without language input.

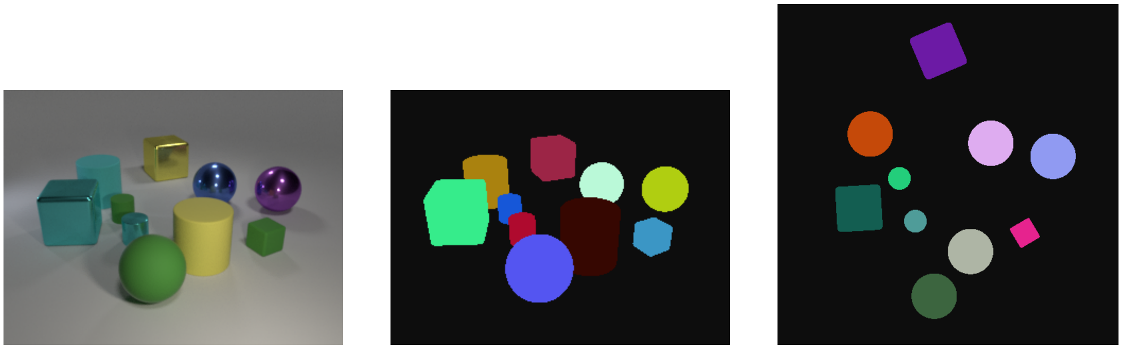
In Section 6.4, we note that some path/command examples convey less information than others because the path changes little, with or without the command. To improve convergence and performance on hard examples, we develop a minibatch selection algorithm to bias sampling towards the most problematic examples and find that it does indeed improve performance. We also experiment with enhancing the gradient information by randomly exploring sub-optimal states adjacent to the lowest-cost path.

Finally, in Section 6.6 we present some initial results with a hybrid dataset, developed with human-labeled aerial images and augmented with synthetic commands. Our results are preliminary, and while the algorithm trains successfully, it does not perform better than our baseline for the validation data. Nevertheless, we are confident that future work will improve upon this result.

6.1 Synthetic Dataset

The synthetic dataset is derived from CLEVR (Johnson *et al.* [25]), a diagnostic dataset developed for early VQA systems such as DDRProg (Suarez *et al.* [75]). The images are computer generated using Blender [130], a professional computer animation tool with realistic rendering capabilities, including specular and diffuse reflections and shadows with multiple light sources (Figure 6.1a). The CLEVR dataset is generated using a script that creates random scenes and questions according to a sequence of templated operations. It is trivial to generate answers with full awareness of the virtual world. Early efforts to solve this dataset focused on predicting the sequence of template operators that were used to answer a question (the program) and then instantiating a modular neural network to produce a solution, such as in Suarez *et al.* [75]. FiLM makes no assumptions about the underlying model which is appropriate for our application.

We heavily modify the data generation scripts from Johnson *et al.* [25] to make a new synthetic dataset for navigation. Figure 6.1 shows one example scenario from the regenerated CLEVR dataset.



(a) Synthetic image (b) Perspective mask (c) Aerial orthographic mask

Figure 6.1: Synthetic image generated with perspective and aerial masks. Each object in the mask is identified by unique coloration.

Table 6.1 shows the basic template for each command, with the meta-tags corresponding to the properties in Table 6.2. We have omitted alternative templates that express the same concept with a different form for brevity. Each template is associated with a program that utilizes the information in the scene graph to randomly generate interesting cost maps that fulfill the command.

Figures 6.2 to 6.6 are examples of the command families from Table 6.1. The cost maps in these figures are generated from testing data using the model trained in Section 6.1.1. This cost map is *not* the cost map generated by the templates, but rather is inferred from the image features, commands, and path data.

The dataset used in Section 6.1.1 has 2048 randomly generated scenes in the train, test, and validation sets. For each scene, 16 paths for each of the five command families are randomly generated, yielding 32768 examples per set.

Table 6.1: Command types in synthetic dataset. Object tags in Table 6.2. Optional properties in parentheses. Alternative templates (not shown) and word substitution generate a wider variety of commands than apparent in this sample. Command family 5 used only in result presented in Table 6.13.

Family	Command Template
0	**Single set of objects** Go ⟨A⟩ing [the] (⟨Z⟩ ⟨C⟩ ⟨M⟩) ⟨S⟩.
1	**Union of two sets of objects** Go ⟨A⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (along with) [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
2	**Intersection of two sets of objects** Go ⟨A1⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ and ⟨A2⟩ing [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
3	**Subset of a set** Go ⟨A⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (except) [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
4	**Choice of object sets** Go ⟨A1⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ or [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
5	**Agent-relative object identification** Go ⟨A⟩ing [the] ⟨Z⟩ ⟨C⟩ ⟨M⟩ ⟨S⟩ ⟨D⟩[est] to [me | my location].

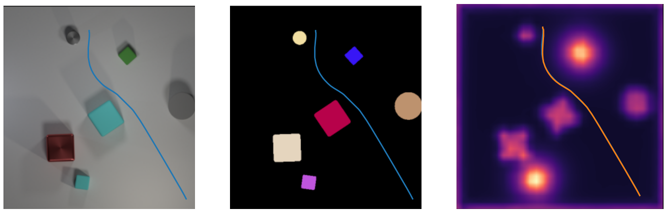
Table 6.2: Object attributes for synthetic dataset and associated template tags.

Description	Tag	Options
Affordance	⟨A⟩	Avoid, Trust
Color	⟨C⟩	Gray, Red, Blue, Green, Brown, Purple, Cyan, Yellow
Distance	⟨D⟩	Near, Far
Material	⟨M⟩	Rubber, Metal
Relation	⟨R⟩	Left, Right, Back, Front
Shape	⟨S⟩	Cube, Cylinder
Size	⟨Z⟩	Small, Large

The synthesized cost map has a resolution of 128×128, and the ground truth paths are scaled accordingly. We generate paths using A*, Field D*, and OMPL (Şucan *et al.* [131]) path planners. The OMPL path is the shortest distance non-obstacle path and represents an agent that avoids obstacles and uses no other information. We pre-compute the Fréchet and Hausdorff metrics for the A* and Field D* paths with respect to this baseline.

Scene images are rendered at 512×512 resolution. This is down-sampled to 256×256 for feature extraction. We use ResNet-101 (He *et al.* [82]) mid-level features which result in $\mathbb{R}^{256\times32\times32}$ feature maps. This model has been pre-trained on the ImageNet V2 dataset (Russakovsky *et al.* [132]) and supplied by the Torchvision package (Marcel *et al.* [133]).

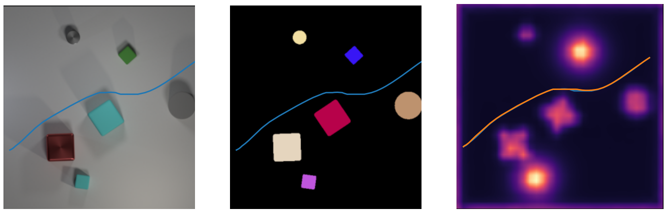
We use ResNet features because the original FiLM algorithm does as well, but it is reasonable to assume that other stem networks may produce features more appropriate to our task. The features are static to reduce network complexity due to hardware constraints and training time. However, Command Family 0: Single set of objects



(a) Aerial image (b) Mask (c) Inferred cost map

Figure 6.2: Command: "Navigate avoiding the small rubber things." Ground truth path in blue, inferred path in red.

Command Family 1: Union of two sets of objects



(a) Aerial image (b) Mask (c) Inferred cost map

Figure 6.3: Command: "Hide from tiny cubes along with the small matte objects." Ground truth path in blue, inferred path in red.

future work may fine-tune the stem network after initialization with a pre-trained model. That features from a network with vastly different purposes work at all for our problem testifies to the ability of deep learning to grasp fundamental concepts in vision that apply to many tasks.

The command text is converted into a token sequence which the FiLM LSTM/GRU reads as one-hot vectors that are embedded and then converted into modulation vectors. We experimented with GloVe ([85]), a pre-trained word embedding, and observed no change in performance. However, because GloVe vectors group words with similar concepts "near" each other, it permits some flexibility when testing with commands that contain words not seen in the training corpus.

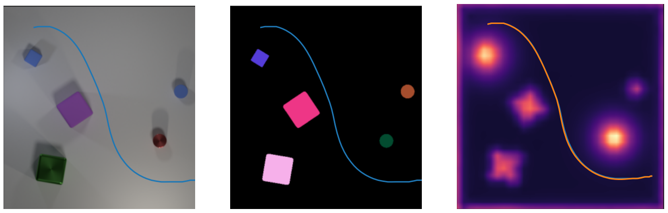
For more detail on the dataset generation process, please refer to Section 8.1

6.1.1 Meta-Parameter Selection

This section introduces a basic model and experiments with the generic meta-parameters to establish a performance baseline. We vary the learning rate (η), regularizer (λ), batch size (N), Huber delta parameter (δ), and optimizer choice. The data presented here come from an early version of our architecture that uses A* instead of Field D* with a simplified command. We have noted that the Field D* variant is more robust to parameter selection than the A* variant, possibly because the gradient from Field D*, while also sparse, is not quantized like the gradient from A*.

The model is sensitive to the selection of the learning rate parameter, η. Large values cause

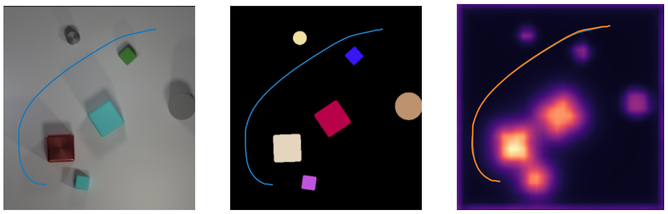
Command Family 2: Intersection of two sets



(a) Aerial image (b) Mask (c) Inferred cost map

Figure 6.4: Command: “Go giving wide berth to metal cylinder and fearing tiny blue matte block.” Ground truth path in blue, inferred path in red.

Command Family 3: Subset of a set



(a) Aerial image (b) Mask (c) Inferred cost map

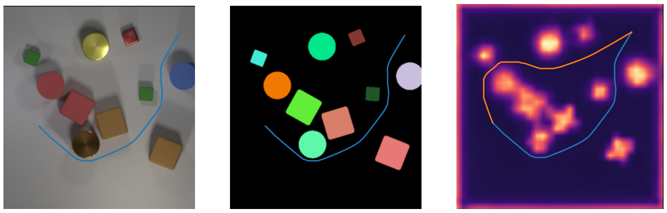
Figure 6.5: Command: “Navigate giving wide berth to cubes except the tiny green block.” Ground truth path in blue, inferred path in red.

numerical overflow instabilities, despite efforts to augment precision. This is especially true in the early part of training but can occur sporadically in later epochs for batch samples with large errors. We have not observed any benefit to training with small learning rates over long periods, as the authors of Perez *et al.* [24] have observed. On the contrary, we have found that small learning rates generally lead to over-fitting and poor validation performance. The dataset used for these experiments is considerably simpler than theirs, and this may yet be a good strategy in future work. Typical values are $\eta \in [10^{-3}, 10^{-4}]$. The model is relatively insensitive to the regularizer parameter, λ, which is typically $[10^{-5}, 10^{-6}]$.

The model is not very sensitive to the value of the Huber δ parameter, the switching point between L_1 and L_2 loss. It is most important for containing extreme losses and gradients in the initial part of the training. We typically use a value between 0.1 and 10 with little notable change in convergence or ultimate performance. The L_2 version of the loss metric generally leads to a lower ultimate error. A smaller value of δ lead to prolonged use of the L_1 loss and higher final error.

We have observed that a batch size between 80 and 128 examples leads to the fastest convergence. Small batch sizes cause instability, likely due to large sample variance. In contrast, large batch sizes lead to slower convergence with a larger ultimate validation error, probably due to mundane examples masking the gradients of more informative ones. These observations are consistent with machine learning literature.

Command Family 4: Choice between sets



(a) Aerial image (b) Mask (c) Inferred cost map

Figure 6.6: Command: "Trust the brown metal object or the brown rubber blocks." Ground truth path in blue, inferred path in red.

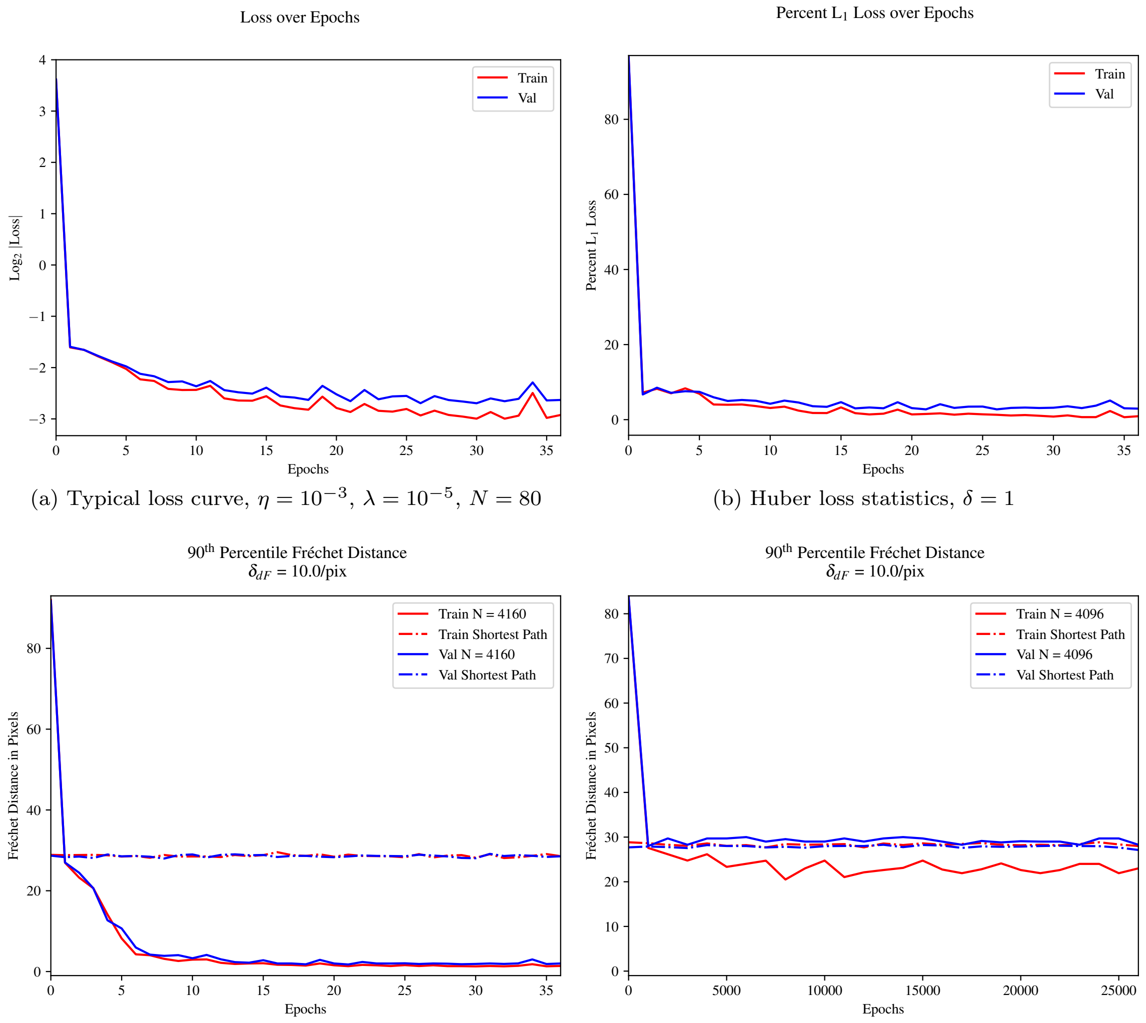
We have experimented with various optimizers (SGD, AdaGrad, Adam, AMSGrad) with no notable difference in training performance.

Figure 6.7 illustrates the behavior of a properly tuned system. The Huber loss (Figure 6.7b) switches from L_1 loss to L_2 loss at approximately the same point where the performance of the system passes the performance of a näive system which chooses the shortest non-obstacle path between the given endpoints. There is a pronounced knee in the loss (Figure 6.7a) that occurs at this point. The Fréchet error (Figure 6.7c) tracks the max-margin loss. Specifically, we monitor the value of the 90th percentile of the Fréchet metric for the 4160 sample paths. The total run-time was 6 hours.

As a comparison, runs with no language input (Figure 6.7d) show performance on par with the baseline metric. This was accomplished by fixing the FiLM parameters to $\gamma = 1$ and $\beta = 0$. Table 6.3 shows the performance of the model at the end of the training run for the different command types, with the scores for the 90^{th} percentile Fréchet distance highlighted. Command family 4 requires reasoning about the starting and ending point of a path and is more challenging to learn. For both the modified Hausdorff metric and the Fréchet distance, the mean is significantly larger than the median, indicating that there are outlier paths that the model does not learn well and are not well represented in the loss. Examples of the worst-performing paths from this run can be found in Section 8.2. We examine the errors in detail in Section 6.2.

Table 6.3: Metrics by command family from validation set, 4160 samples.

	Loss				Fréchet				Modified Hausdorff			
Family	*Mean*	*Std Dev*	*Median*	*90th Pct*	*Mean*	*Std Dev*	*Median*	*90th Pct*	*Mean*	*Std Dev*	*Median*	*90th Pct*
0	1.7e-03	2.7e-03	1.4e-03	2.1e-03	1.34	5.13	0.57	1.05	0.42	2.12	0.15	0.36
1	1.7e-03	1.8e-03	1.4e-03	2.2e-03	1.35	4.35	0.66	1.17	0.43	1.88	0.16	0.40
2	1.9e-03	2.3e-03	1.5e-03	2.6e-03	1.83	6.19	0.70	1.19	0.65	2.80	0.17	0.46
3	1.7e-03	1.3e-03	1.4e-03	2.3e-03	1.59	6.08	0.64	1.09	0.57	2.89	0.17	0.41
4	3.0e-03	2.7e-03	2.3e-03	5.1e-03	3.74	7.30	1.28	8.25	1.61	3.78	0.48	2.98



(a) Typical loss curve, $\eta = 10^{-3}$, $\lambda = 10^{-5}$, $N = 80$

(b) Huber loss statistics, $\delta = 1$

(c) 90th percentile Fréchet loss

(d) 90th percentile Fréchet loss, no language input

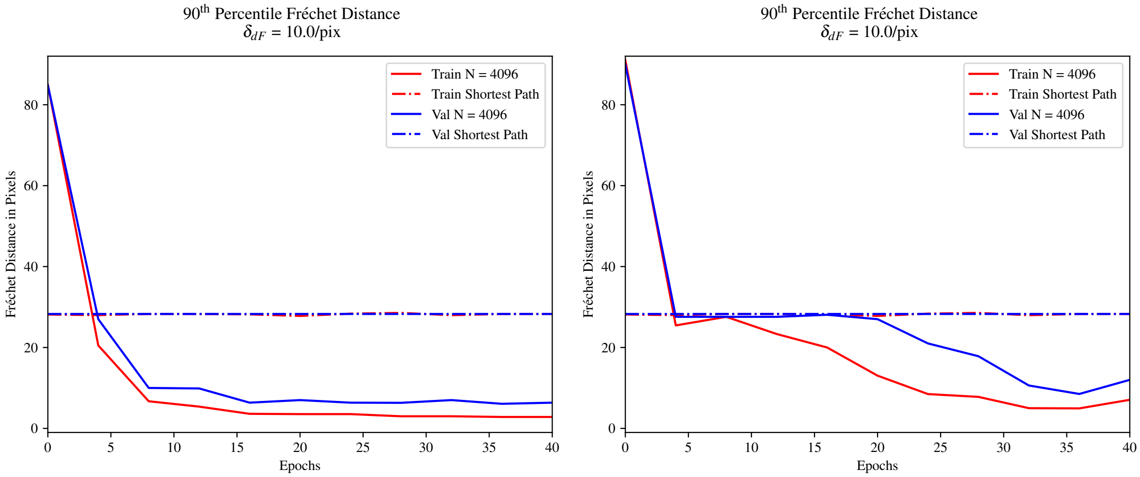
Figure 6.7: Typical training performance plots for a properly tuned system. Baselines in Figures 6.7c and 6.7d (dashed) are Fréchet distance between non-obstacle shortest paths and ground truth paths. Figure 6.7d shows that language is essential to perform well on dataset.

6.1.2 Output Function

We experimented with different output functions for the network. Both Field D* and A* do not work with negative values as these can induce infinite cycles. In some of our early experiments, we avoided this problem by prohibiting states from being visited more than once. This can still lead to problematic results for the loss function when computing path gradients and is only suitable for cases where negative values occur in only the minority of states.

A more durable solution is to use a non-negative output function. Viable choices are the absolute value, quadratic, or sigmoid functions. The sigmoid is a poor fit for our task since there is a rapid change in the derivative between the two saturated regions by design. Ratliff [58] use an exponential function in their LEARCH algorithm. Networks with these output functions are volatile in the initial epochs.

In general, when using the quadratic output function, we see a foot in the performance plot (Figure 6.8b), where training slows for a bit. This foot occurs approximately where the Huber loss transitions from L_1 to L_2 loss. In general, networks with the absolute value output function (Figure 6.8a) train more quickly, with validation performance tracking training performance more closely. With the quadratic output function, there is a lag of at least ten epochs between training and validation performance. Generally, there is a much slower convergence rate with results no better than for the networks with the absolute value output.



(a) Fréchet metric, absolute value output function

(b) Fréchet metric, quadratic output function

Figure 6.8: Performance metrics for networks with absolute value function and quadratic output function.

To understand the reason for this, consider the loss function as defined in Equation (6.1), where q is either 1 or 2. If $F(\cdot)$ is a quadratic function of the network output, and $q = 2$, then we are effectively optimizing over the 4^{th} power of the network output, which likely creates extreme values and gradients that affect numerical performance.

$$L = \min_{\theta} \left(\frac{1}{N} \sum_i \frac{\beta_{ij}}{q} \left\| F(\mathcal{I}_i, Z_j; \theta)\mu_{ij} - \min_{\mu \in \mathcal{G}_i} \left(\underbrace{F(\mathcal{I}_i, Z_j; \theta) + \ell(\mu_{ij})^T}_{\text{loss-augmented cost map}} \right) \mu \right\|^q + \frac{\lambda}{2} \|\theta\|^2 \right) \tag{6.1}$$

6.2 Error Analysis

In most cases, the ground truth cost map is not expressible or knowable for human-labeled data. One benefit of having a fully synthetic dataset is that we have access to the ground truth cost maps.

In the following section, we use this information to lend insight into the kinds of errors made by the algorithm. Our analysis determines whether paths with similar costs yet largely different trajectories contribute significantly to the aggregate error. Max-Margin Planning cannot model examples with multiple correct solutions, unlike Max-Entropy techniques that model the distribution over solutions.

We generate a cost map and path, $\hat{\mu}_i$, for each input $(\mu_i, \Lambda_i, \mathcal{I}_i)$. We compute the path cost for $\hat{\mu}_i$ on the ground truth cost map, which is never otherwise used for testing or training and is usually discarded during dataset generation. We then compute the path cost for μ_i but on the ground truth cost map. For each example, we compute the Fréchet distance between paths and the percent error in the cost relative to the ground truth cost. The paths in this analysis are generated with the A* algorithm.

We use the ground truth cost map for this analysis because the generated cost map is not guaranteed to have the same scale as the original. Although unlikely, it is also possible that the generated cost map does not resemble the latent cost map because MMP is a discriminative classifier. It is sufficient for it to create paths that imitate the training data. It is also not strictly possible to compare the cost maps directly.

Figures 6.9a and 6.9c are 2D histograms of the percent cost error relative to ground truth cost vs the Fréchet distance metric. The count/coloration of the plot is on a logarithmic scale. It shows that a large number of paths have large Fréchet distances to ground truth yet have *low* cost (left border). This confirms that a significant fraction of the Fréchet loss derives from alternative paths that have almost equal cost to the ground truth.

Figures 6.9b and 6.9d contain the same data as the prior plots with a logarithmic x-axis.[1] The training set (Figure 6.9b) can be partitioned into four regions.

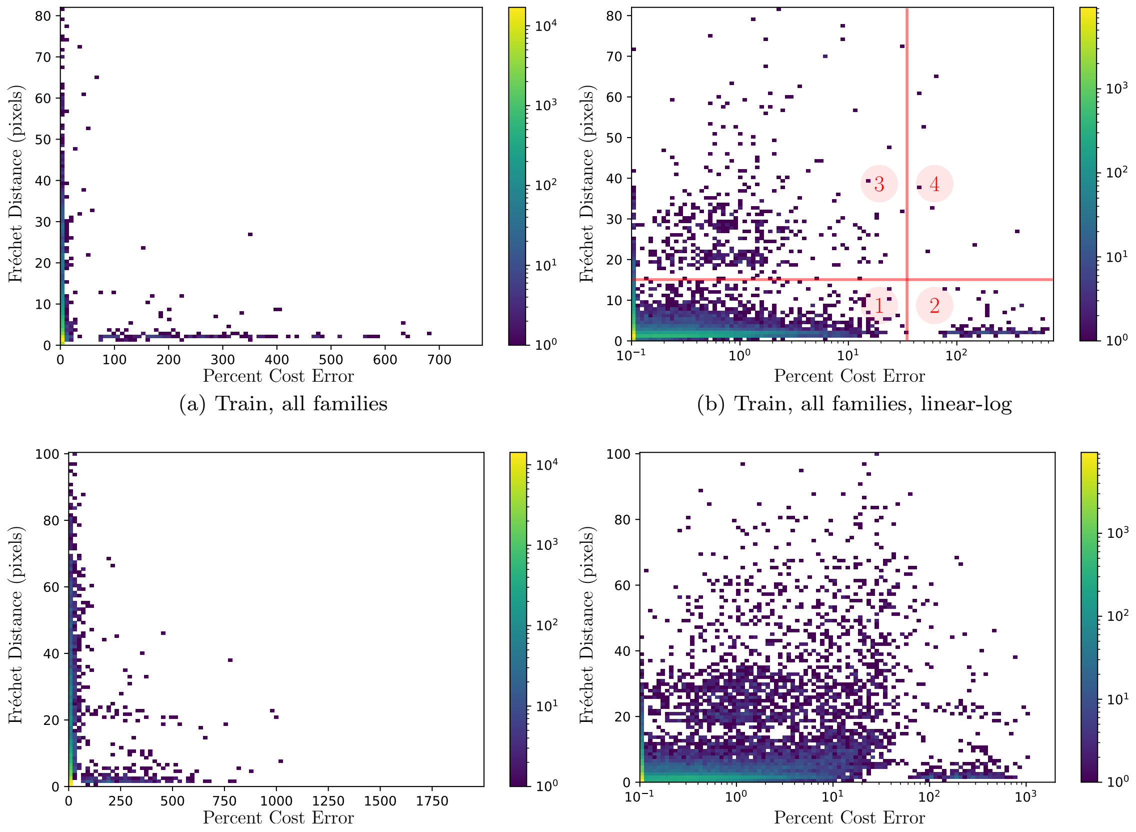
- **Region 1** is the set of paths with low Fréchet distance and low path cost error. These paths approximately match the ground truth paths and have “normal” errors.
- **Region 2** is the set of paths with high cost but low Fréchet distance.
- **Region 3** is the set of alternative paths with a similar cost to ground truth but with high Fréchet cost.
- **Region 4** contains those paths with a high relative cost error and high Fréchet distance.

We have selected the worst examples from each region to visualize the kinds of errors the algorithm produces. Please refer to the appendix in Section 8.3 for the following discussion.

Region 2 paths (Section 8.3.2) typically cut through an object and incur high costs. Examining the ground truth and learned cost maps in Figures 8.9b and 8.9c, we see that the learned cost map lacks sharp object boundaries. In general, it is known in robot navigation that non-traversable regions are hard to enforce with a cost map alone. While one could use infinite costs to designate undesirable areas, this may cause scaling problems. Explicit keep-out regions are a better solution and point towards future work to make the algorithm practical.

Region 3 generated paths (Section 8.3.3) have similar costs to the ground truth path but take very different trajectories. The generated tracks have comparable lengths to the ground truth path,

[1] Cases where the percent cost error is $< 10^{-1}$ are assigned a value of 10^{-1} to avoid $\log(0)$.



(a) Train, all families

(b) Train, all families, linear-log

(c) Test, all families

(d) Test, all families, linear-log

Figure 6.9: Error histograms for train and test dataset, all families.

and the generated cost map is similar to the ground truth cost map. The difference in cost between the paths is less than 15 percent. The algorithm did produce a path of similar cost to the expert if measured by the ground truth cost map.

Region 4 errors (Section 8.3.4) are paths with large cost error and large Fréchet error. This is usually the result of a clearly incorrect cost map. Figure 8.19 shows that the learned cost map incorrectly increased cost around the red cylinder on the left (perhaps because it looks brown) and takes a longer path to the right of the image. Figures 8.21 to 8.23 seem to overestimate the cost around objects that are to be avoided, forcing the planner to choose a longer route that is more costly on the ground truth cost map. Figure 8.20 seems to fail similarly, but it also incorrectly increases the cost for small purple objects. This error increases the cost for the learned path but does not increase the cost for the ground truth path, indicating the cost in the vicinity of the ground truth path must be overestimated.

Section 8.4 contains supplementary error histograms for each command family, excluding command family 4. These plots largely echo the patterns observed in the aggregate data.

Table 6.4 gives statistics for each region and how they affect the Fréchet score of the classifier.

Table 6.4: Count of datapoints in each region, 90th percentile Fréchet distance metric for accumulated regions.

	Region	Count	$^{90}F_{GT}$	Region
Train	1	39138	4.00	1
	2	197	4.00	1,2
	3	625	4.47	1,2,3
	4	18	4.47	1,2,3,4
Test	1	37429	5.10	1
	2	501	5.66	1,2
	3	1591	7.00	1,2,3
	4	392	7.78	1,2,3,4

Table 6.5: Frequency of each command family in error regions 2, 3, and 4.

	Family	0	1	2	3
Region 2	Train	0.127	0.365	0.310	0.198
	Test	0.076	0.343	0.411	0.170
Region 3	Train	0.189	0.318	0.246	0.246
	Test	0.206	0.300	0.275	0.219
Region 4	Train	0.056	0.389	0.556	0.000
	Test	0.023	0.472	0.398	0.107

The Fréchet metric is not significantly affected with the addition of region 2. These high-cost paths still have low Fréchet distance. Including region 3 increases the Fréchet loss notably. The addition of region 4 error does not contribute to the overall loss as significantly.

Table 6.5 shows the frequency that each command family appears in either region 2, 3, or 4. Command families 1 and 2 are the most significant error source.

Table 6.6 shows the Fréchet score for each command family for both the generated path ($^{90}F_{GT}$) and the baseline shortest non-obstacle path ($^{90}F_{SP}$). In all cases $^{90}F_{GT}$ is less than the naïve shortest path solution, $^{90}F_{SP}$. Additionally, we compare the distributions of the generated paths' Fréchet metric to that of the baseline by using the Kullback-Leibler divergence to show that the distribution is quite different (which is good).

However, for family 4, performance for validation and testing is almost on par with the baseline. This is because commands of this variety cannot be expressed with just an image and command but also require path endpoints as input to the network. This was not provided to this model. This command class's relatively good training performance can be attributed to over-fitting.

6.2.1 Summary

We have decomposed the error into four regions by regenerating the ground truth cost map and re-analyzing the paths generated by the algorithm. This procedure is only possible with synthetically

Table 6.6: 90th percentile Fréchet distance in pixels ($^{90}F_{GT}$) for each command family compared to ground truth (lower is better) and shortest non-obstacle path baseline ($^{90}F_{SP}$) compared to ground truth. KL divergence (D_{KL}) between distribution of Fréchet distance to ground truth (p) and non-obstacle shortest path distribution to ground truth (q), in bits. Higher is better.

		Train			Validation			Test	
Family	$^{90}F_{GT}$	$^{90}F_{SP}$	D_{KL}	$^{90}F_{GT}$	$^{90}F_{SP}$	D_{KL}	$^{90}F_{GT}$	$^{90}F_{SP}$	D_{KL}
All	4.95	29.35	4.60	10.61	30.00	3.58	10.61	29.67	3.58
0	4.00	28.28	4.89	5.10	28.43	4.40	5.00	28.28	4.04
1	4.95	31.20	4.59	9.90	32.57	3.25	9.90	31.82	3.72
2	5.00	31.00	4.69	9.00	32.03	3.43	9.00	31.31	3.30
3	4.24	30.38	4.76	6.08	30.74	4.34	6.00	30.08	4.11
4	5.00	27.20	4.31	24.75	27.83	1.97	24.75	27.58	1.82

generated data since it is not, in general, possible to express or even know the cost map a human uses to make similar decisions.

As evident from region two paths, the algorithm may benefit from a separate representation for non-traversable areas. While this affects only about 1 percent of paths, a better cost map representation may improve performance for all examples by mitigating potential numerical problems from representing very high-cost forbidden regions concurrently with low-cost traversable areas.

In region three paths, we see how MMP has learned to generalize the training examples into cost maps that correctly generate paths of similar cost. As noted in the literature, MMP can not handle cases where multiple paths have equal costs. This behavior is expected and does not result in pathological outcomes. The algorithm seems to converge towards sensible cost maps when trained on aggregate data. We suspect that the erroneous examples in region three are local minima that approximate the original cost map but are very hard to escape. Switching to an algorithm like MaxEnt IRL may be one way to eliminate errors of this variety, but with added numerical complexity.

In region 4, we see that the number of genuinely wrong examples is relatively sparse, about 1 percent in the testing set. While we cannot offer a specific remedy to these examples, it is worth exploring how changing meta-parameters and network architecture affects these outliers.

6.3 Visualizing the Effects of Language

In this section, we show how language input affects the cost map. We can see that the algorithm appropriately modulates the cost map by examining it with and without language input. We also perform a clustering experiment on the modulation vectors to visualize the hidden structure learned by the model.

	Typical Command
Command	Navigate giving wide berth to gray object
Tokens	[<START>, navigate, giving, wide, berth, to, gray, object, <END>]
Encoded	`[ 1 46 31 68 12 64 33 48 2 0 0 0 0 0 0 0 0 0 0 0 0 ]`

	Empty Command
Command	-
Tokens	[<START>, <END>]
Encoded	`[ 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]`

Table 6.7: A typical command and an empty command transformed into tokens.

6.3.1 Cost Map Modulation

We use a similar model to the one used in Section 6.2. We extract the cost map for a given command in the usual fashion and compare it against a cost map without language input. We use the same model to generate this cost map to make the comparison as direct as possible. Direct comparison is only possible after accepting two approximations.

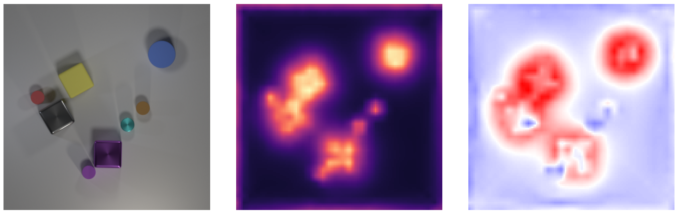
First, we assume that two cost maps will have a similar scale. Cost maps are not unique. For example, increasing the cost of an object can have a similar effect as reducing the costs of others. Nonetheless, we assume that we can directly compare one cost map to another. In Section 6.2, we carefully avoided this assumption.

Second, we require modulation vectors for the feature maps that are comparable with those upon which the model was trained because the FiLM architecture in Figure 3.9 uses the feature modulation vectors to generate a cost map. We, therefore, devise a procedure to elicit modulation vectors from the language encoder that contain no command content. It is important to note that the architecture is not being re-trained with empty commands. We presume that it has generalized enough that empty commands (which have not been trained upon) will produce neutral cost maps. Directly setting the modulation vectors' $\gamma = 1$ and $\beta = 0$ does not produce usable cost maps.

Each command is encoded into a string of numerical tokens (Table 6.7), with three special tokens: `<START>`, `<END>`, and `<NULL>`. `<NULL>` tokens pad the end of a token stream so that all commands have the same length. The tokens are embedded with word2vec. The hidden state of the RNN is extracted after the `<END>` token. That hidden state ($\mathbb{R}^{7168}$ in this example) is transformed into the feature modulation vectors described in the prior section using a fully connected layer. The empty command is simply the `<START>` and `<END>` tokens which produces modulation vectors that generate plausible cost maps without language content.

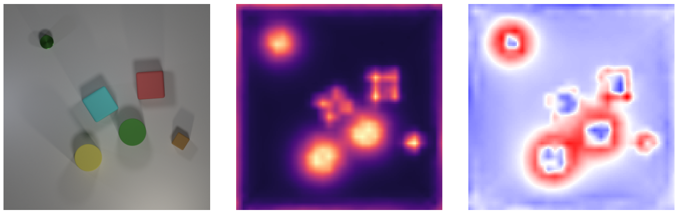
We present 20 random examples drawn from the validation set here and in Section 8.5 to illustrate how the language modulates the cost map. Figures 6.10, 6.11, 6.13, 6.14 and 6.19 are good examples of the 5 commands families.

Figure 6.16 is an example of a poor result. The algorithm detects none of the referent objects. Inspection of the generated path (not shown for clarity) shows that it has a low Fréchet distance



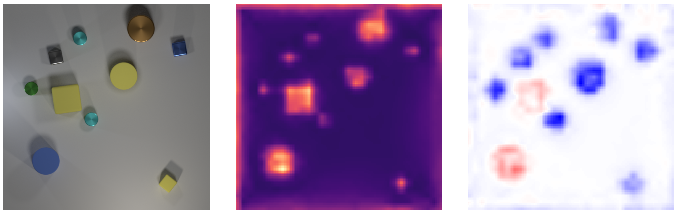
(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.10: Command: “Fear big things.” Command family: 0.



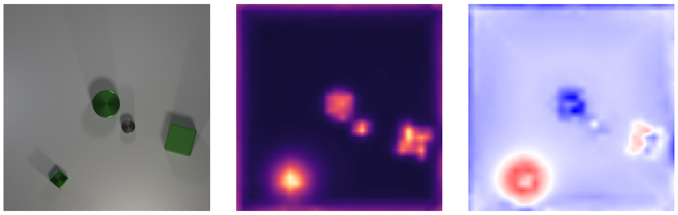
(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.11: Command: “Proceed avoiding the large matte cylinders including the tiny shiny object.” Command family: 1.



(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.12: Command: “Navigate trusting the small objects including the yellow rubber cylinder.” Command family: 1.



(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.13: Command: “Stay away from small cube and fear the small green block.” Command family: 2.

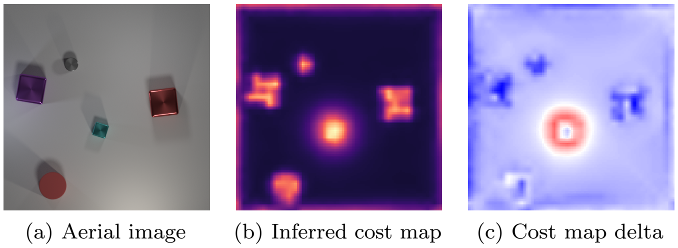

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.14: Command: "Avoid shiny cubes except for the large cubes." Command family: 3.

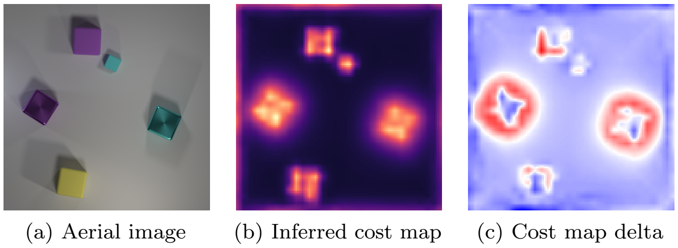

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.15: Command: "Stay away from the large cubes but not matte things." Command family: 3.

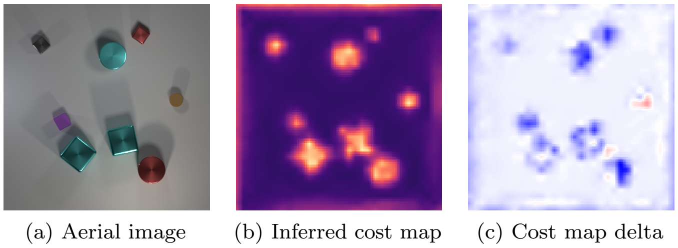

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.16: Command: "Proceed trusting the purple cube or the tiny red shiny cube." Command family: 4.

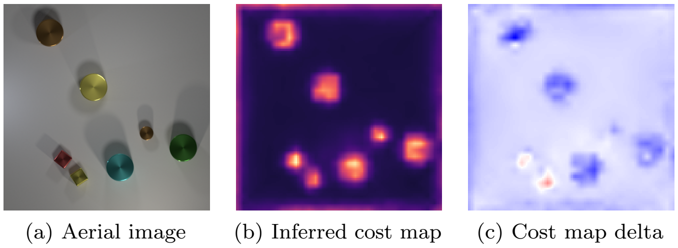
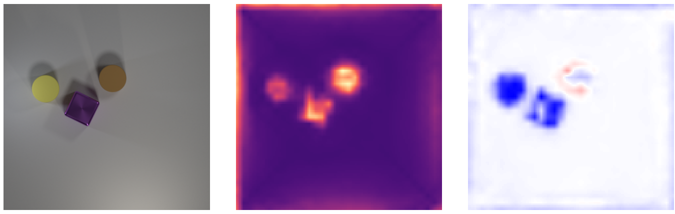
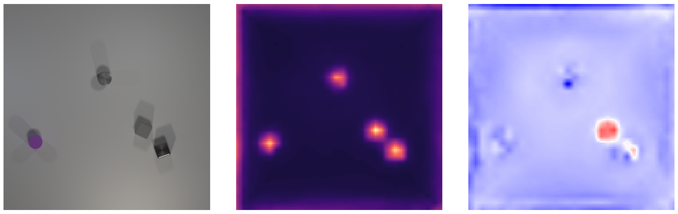

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.17: Command: "Proceed keeping away from the big cyan metallic object or big brown cylinder." Command family: 4.



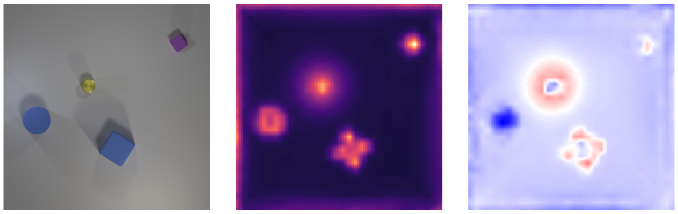
(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.18: Command: "Proceed trusting the yellow rubber object or the large metal cube." Command family: 4.



(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.19: Command: "Go hiding from the purple matte object or tiny gray cylinder." Command family: 4.



(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 6.20: Command: "Avoid the big blue matte cylinder or tiny shiny cylinder." Command family: 4.

from the ground truth path *despite* not correctly modulating the cost map.[2]

It is possible that the way we approximate the no-language output has introduced an error that is of no consequence to actual testing scenarios. However, this is also an example of a command that is inconsequential to the path and, therefore, a low-information training example as evidenced by the small Fréchet error between the ground truth and shortest paths. The algorithm will not learn how language and image features map to a proper cost map given low-quality training samples similar to this one. How to measure the information in the training example and weight them

[2] Command family 4 is has poor performance, but this is expected since identifying the cheapest of two plans requires knowledge of the starting and stopping points of the path. This was accidentally omitted when performing the analysis.

accordingly is an interesting problem that we address in Section 6.4.

6.3.2 t-SNE Clustering of Language Encoder Output

In this section, we perform a clustering analysis on the FiLM modulation vectors to gain insight into what the algorithm learns. We used an accelerated version of t-SNE by Ulyanov [134], which was originally published by Maaten *et al.* [135] and Maaten [136].

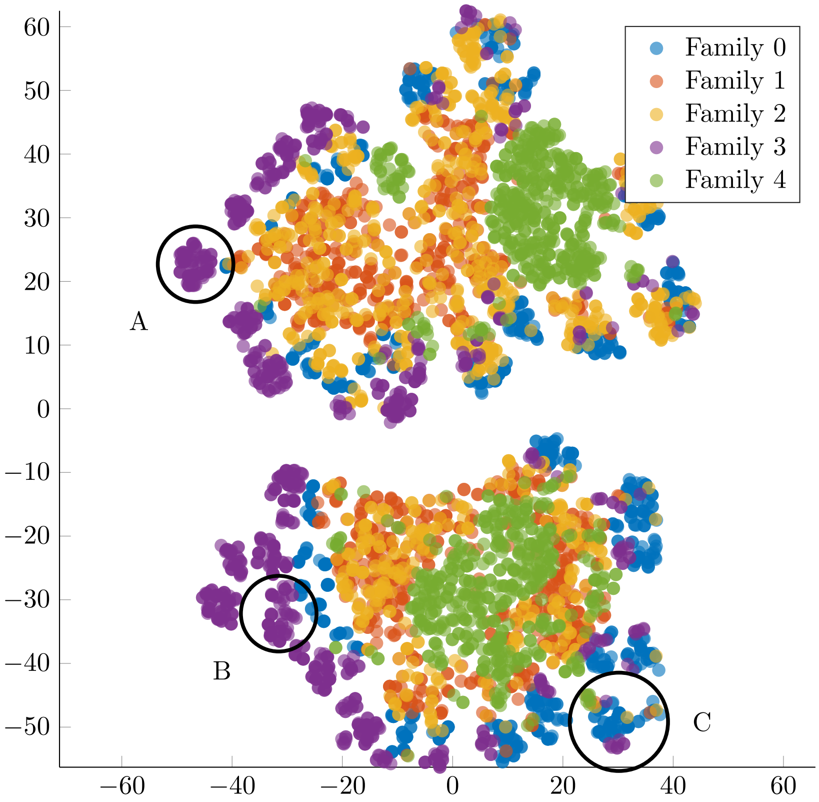


Figure 6.21: t-SNE plot for all command families, training set. Perplexity = 50, $N = 5000$.

Figure 6.21 shows clustering of 5000 FiLM vectors from the training set derived from the hidden state of the language encoding RNN. The vectors are $\mathbb{R}^{1024}$ and modulate the feature maps that generate the cost map by scaling and offsetting each channel. Note how the points corresponding to command family 4 (green) are segregated from other points. These correspond to commands that offer a choice; “Avoid the red cube or avoid the blue cylinder.” They are fundamentally different than the other commands in that the algorithm must make an inference about which option set produces a cost map and path with the least cost.

The purple points are also segregated into smaller clusters. Command family 3 has commands of the form “Trust the blue cubes except for the small ones,” which exercise computing the subset of a set. A deeper inspection of the individual points in each cluster reveals that they contain commands with similar structures. For example, the cluster marked “A” has commands that all begin with “Avoid the objects except...” The cluster marked “B” has commands that trust various kinds of cube shapes.

This pattern appears to be true for command family 0 as well, which has the form “Go fearing the blue cubes”. In general, we observe that clusters generally form from commands that refer to similar object types. The cluster marked “C” has commands that all refer to various cyan objects, even for data points in commands family 3 (purple).

Commands families 1 and 2 are generally commingled. Command family 1 has the form “Go [fear | avoid] the [objects] and [other objects].” Command family 2 has the form “Go [fear | avoid] the [objects] and [fear | avoid] [other object].” Upon closer inspection of the template used to generate these two commands families, it is apparent that the algorithm has correctly identified that commands family 1 as a subset of 2.

Figure 6.22 illustrates a similar pattern in the validation dataset.

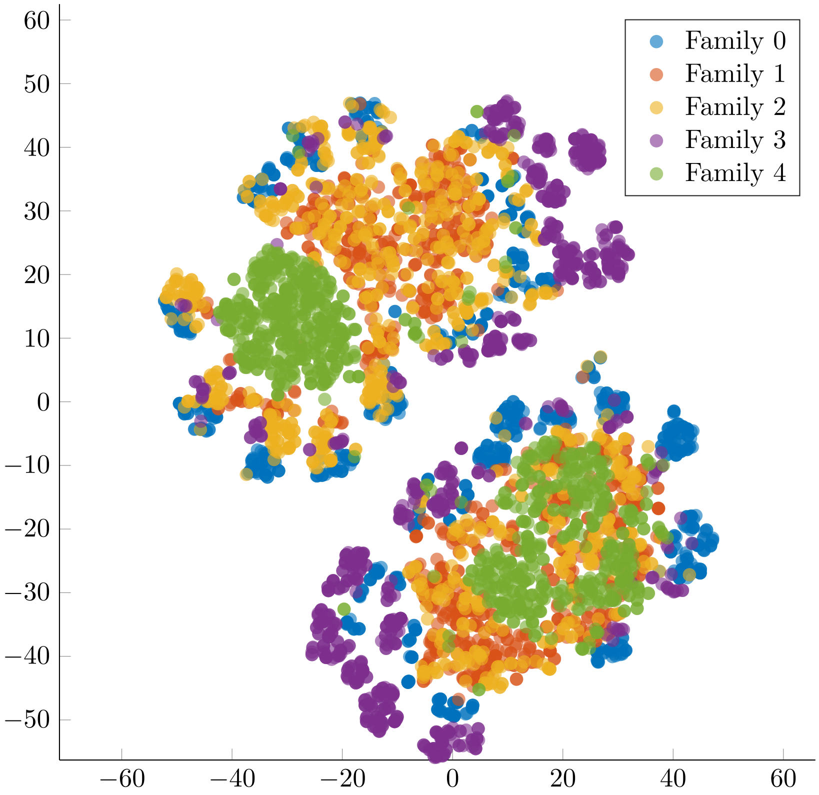


Figure 6.22: t-SNE plot for all families, validation set. Perplexity = 50, $N = 5000$.

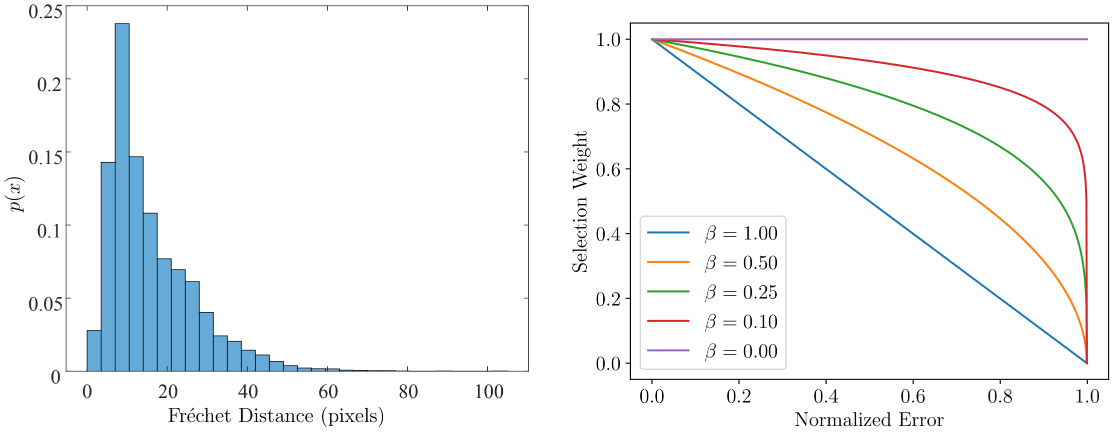
The clustering analysis lends insight into what the algorithm is learning. However, it is important to note that while t-SNE is a valuable tool to visualize high-dimensional data, the results are stochastic with each run of t-SNE, just as the data are stochastic with each training run. The distance between clusters does not indicate how well the algorithm separates commands expressing different concepts from each other. It would be interesting to see how this clustering analysis changes with more comprehensive datasets and elaborate commands.

6.4 Learning from the Most Informative Paths

In Section 6.3.1, we noted the existence of low-information training examples. In this section, we analyze our dataset to determine the extent of this problem and offer a revised training procedure to mitigate the effects.

Low-information commands are those for which the ground truth path differs little from the non-obstacle shortest path between endpoints. The algorithm learns the meaning of a command by observing its impact on the ground truth trajectory. While commands with trivial answers are an important part of learning, too many such commands can also interfere with learning by making default strategies unreasonably successful.

Figure 6.23a depicts the distribution of the Fréchet distance of paths with respect to the baseline shortest non-obstacle path. More than 50 percent of the paths are very similar to a naïve plan, indicating that those paths provide relatively little information to the learning process. For example, a command "Avoid the red cubes" does not alter a path if there are no red cubes or if they are so far away to have little effect.



(a) Distribution of Fréchet distance between ground truth and non-obstacle shortest path for CLEVR-derived dataset.

(b) Sampling weight curves for varying β.

Figure 6.23: More than 50 percent of all paths have a Fréchet distance of fewer than 10 pixels when compared to the shortest path, regardless of command (left). Example error histogram (right, $\beta = 1$, blue line) used to generate new mini-batch selection weights.

The imbalanced class learning problem in object classification literature arises from long-tail distributions with many rare object classes that pose a significant challenge to learning. One solution to the problem is to statically rebalance the dataset by removing examples with trivial solutions so that the network trains on a more diverse set of problems. The original CLEVR dataset generation code from Johnson *et al.* [25] rejects questions that have trivial answers with high probability.

Identifying and rejecting trivial commands and paths is more difficult in our setting. A command may generate a relatively direct path after balancing many complex constraints. A synthetic benchmark allows us to view this otherwise latent process, as in Section 6.2, but that is not possible for human-labeled data.

Another solution is to sample the dataset dynamically. Jiang *et al.* [137] generate a cumulative distribution function of the losses in each epoch, exponentiated by a selectivity parameter $0 < \beta \leq 1$, to generate a selection distribution. When $\beta = 0$, all examples are uniformly selected for backpropagation. When $0 < \beta < 1$, examples from the long tail are more likely to be selected than otherwise. Figure 6.23b depicts this process for a CDF linearly decreasing error ($\beta = 1$, blue). Jiang *et al.* [137] show they can accelerate training convergence by favoring challenging examples for back-propagation. All examples go through the forward pass, but not all gradients go through the expensive backpropagation step.

Dong *et al.* [138] addresses the class imbalance problem by introducing a class rectification regularizer that rebalances the cross-entropy loss used in their image attribute recognition problem. As with Jiang *et al.* [137], a CDF is constructed for each training batch but sorted by ground truth attributes instead. Those examples in the upper 50^{th} percentile are considered minority classes, and those in the lower 50^{th} percentile are majority classes. This information is used to construct regularizers based on contrastive loss, triplet ranking loss, and distribution similarity. It is not clear how this strategy can map to high dimensional outputs such as paths, where each command may have multiple paths of equivalent cost.

Zhang *et al.* [139] use a repulsive point process to bias minibatch sampling, while Katharopoulos *et al.* [140] and Johnson *et al.* [141] apply importance sampling to deep learning. All three of these methods use the loss function as a proxy for the importance of a sample. Our loss may underestimate the true loss dramatically because it is based on the Hausdorff metric more suited towards points clouds in comparison to the Fréchet metric. These techniques are also more complex than required for an initial exploration of the benefits of dynamic sampling for our problem.

We have adapted Jiang *et al.* [137] as a dynamic mini-batch sampling strategy (Algorithm 19). It is simpler than more principled sampling strategies such as importance sampling. It does not require explicit knowledge of the loss function, as with Dong *et al.* [138], yet by observing the Fréchet error, the strategy increases the likelihood of training on difficult examples. Different from Jiang *et al.* [137], we use the selection process to create mini-batches rather than select loss-gradient components. We use Adam, AMSGrad, and AdaGrad optimizers which record loss statistics over time. Selective back-propagation is more complicated to implement with these optimizers than with SGD. Additionally, constant-sized mini-batches are best suited for efficient GPU utilization. As with Jiang *et al.* [137], we create a histogram and convert the error to a CDF, which is used to generate sample weights, parameterized by a selectivity index, β. We draw samples from the weighted examples with replacements to generate mini-batches for the following epochs.

For the examples in Figure 6.24, we profile the dataset every two epochs and record the Fréchet distance to the ground truth path as an indicator of difficulty. It shows that selectivity with $\beta = 0.25$

Algorithm 19 Dynamic mini-batch sampling for accelerated learning. Parameters are the same as Algorithm 4 with the addition of the profiling interval S, the selectivity parameter $\beta \in [0..1]$, and epochs, which define the number of epochs to execute.

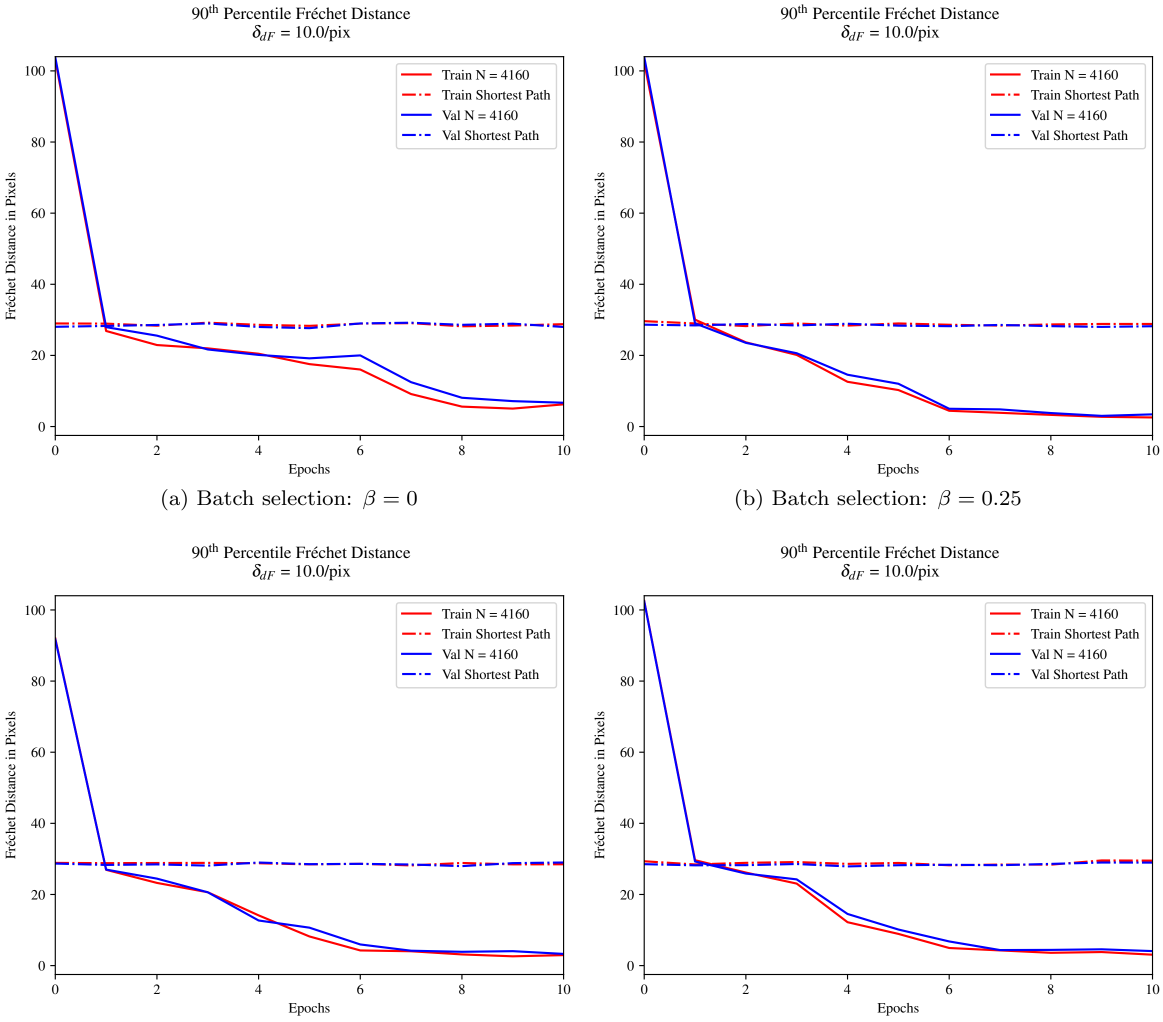
```
function DynamicSampling($\mathcal{I}$, $\mu$, $\Lambda$, $q$, $\delta$, $N$, $S$, $\beta$, epochs)
    $w \sim$ Uniform($\cdot$)                                    ▷ Random initial model weights
    $p(x) \leftarrow$ Uniform($\cdot$)                               ▷ Uniform sample weights
    for $i = 1$ to epochs do                                     ▷ Iterate over epochs
        for $k = 1$ to $\lfloor \frac{|\mu|}{N} \rfloor$ do                                ▷ One epoch
            for $n = 1$ to $N$ do
                $\mathcal{I}_n$, $\mu_n$, $\Lambda_n \sim p(x)$                          ▷ Sample mini-batch
            end for
            $L$, $dLdF \leftarrow$DeepMMPHuberLoss($w$,$\mathcal{I}_N$,$\mu_N$,$\Lambda_N$)          ▷ Forward
            $w \leftarrow$Backward($L$,$dLdF$)                                   ▷ Backward
        end for
        if $i$ mod $S = 0$ then                                    ▷ Profile dataset
            for $j = 1$ to $|\mu|$ do
                $M_j \leftarrow$Forward($w$,$\mathcal{I}_j$,$\mu_j$,$\Lambda_j$)
                $\hat{\mu_j} \leftarrow$Planner($M_j$,$\mu_j$)             ▷ Plan with non-augmented cost map
                $F_j \leftarrow$FrechetMetric($\hat{\mu_j}$, $\mu_j$)                 ▷ Accumulate metrics
            end for
            $P(x) \leftarrow$CDF($F$)                       ▷ Histogram and compute CDF metrics
            $p(x) \leftarrow P(x)^{\beta}$                            ▷ Update selection distribution
        end if
    end for
end function
```

converges faster than training with no selectivity. However, higher selectivity, with $\beta = 0.5$, does not improve convergence further. This suggests a limit to how much this technique can improve training and is likely dataset-dependent.

Table 6.8: ${}^{90}F_{GT}$ for individual command families using mini-batch sampling with various β on validation set. All results from epoch 10.

	Command Family				
β	0	1	2	3	4
0.00	2.00	8.26	10.51	4.38	12.18
0.25	1.16	2.98	3.70	1.33	13.37
0.50	1.09	3.55	3.27	1.56	14.46
1.00	1.17	4.49	4.56	2.00	11.83

Table 6.8 breaks down the performance of the mini-batch sampling technique by command family. Initially, this was envisaged to improve the performance of command family 4, which does not perform as well as other command families. While the performance of command family four has not improved, it is clear that the model learns all the other command families more efficiently than the default sampling strategy when $\beta = 0$. The result in Figure 6.7 use $\beta = 0.50$.



(a) Batch selection: $\beta = 0$

(b) Batch selection: $\beta = 0.25$

(c) Batch selection: $\beta = 0.50$

(d) Batch selection: $\beta = 1.0$

Figure 6.24: Performance for batch selection for single runs. $N = 80$, $\eta = 10^{-3}$, $\lambda = 10^{-5}$.

Practical Considerations

Shuffling and sampling training data on the GPU causes memory fragmentation in our implementation. The GPU runs out of contiguous memory despite sufficient memory, triggering out-of-memory errors. We have adopted the FairScale[142] library to mitigate this, which implements the ZeRO-3[143] and Sharding[144] optimizations for PyTorch. In addition to improved stability, this optimization provides a speed increase for dual GPU training of about 20 percent, along with more usable memory for larger models or batch sizes.

6.5 Epsilon-Greedy A*

The gradient generated by A* is sparse and quantized, corresponding to the eight directions a path may take when crossing a map cell (Figure 4.1). The sparse gradient information may lead to coarse cost maps with low training loss but do not generalize well. We introduce an epsilon-greedy strategy for use with A* path planning during training that generates more diverse gradients by randomly deviating from the ideal policy, similar to Mnih *et al.* [56] as applied to Q-learning. This is not used for testing or collecting runtime metrics.

The problem with sparse gradients is analogous to the exploration-exploitation trade-off in reinforcement learning. A typical solution to this problem is for the agent to follow the optimal policy at every time step with probability $1-\epsilon$, and perform a random action with probability ϵ. The learning algorithm can develop a complete model of the value/cost of policies in the neighborhood of the optimal policy and escape local minima through this strategy.

Epsilon-greedy A* uses similar logic to generate paths close to the optimal path to generate more diverse gradients and explore the vicinity of the optimal policy. By entering sub-optimal states, we gain information about bad strategies that, for example, enter into the interior of objects and would not otherwise be visited.

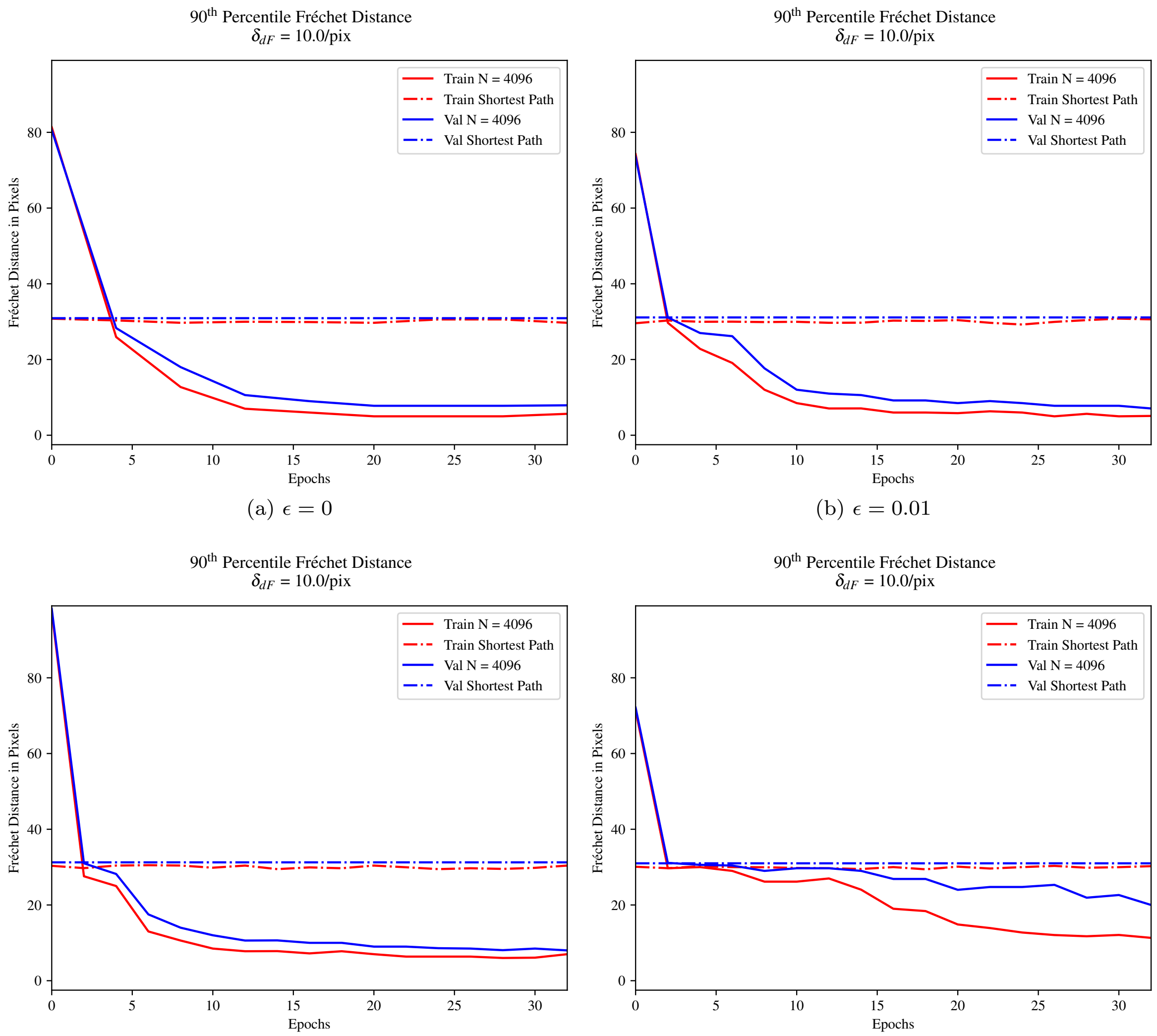
When reconstructing the path from the cost-to-goal table, we choose a random next state with ϵ probability. We carefully avoid cells previously visited to preclude loops. It is necessary to compute the cost-to-goal for all map cells because of the random nature of the path recovery strategy. Therefore, we run A* to completion instead of terminating early. Although improbable, the path can visit every state on the map. If $\epsilon = 0$, the algorithm returns the optimal path as usual.

Table 6.9: Epsilon-A* performance by family for various ϵ on validation set. All results from epoch 32.

	Command Family			
ϵ	0	1	2	3
0.00	6.00	10.24	9.00	7.00
0.01	5.00	10.67	9.00	5.66
0.05	5.66	11.00	9.90	6.00
0.10	11.00	23.64	21.86	21.06

Table 6.9 shows that the performance of the ϵ-greedy A* algorithm is similar to the standard algorithm ($\epsilon = 0$), except for $\epsilon = 0.10$. At this point, random behavior overwhelms the ability to learn, and the algorithm adopts a default non-obstacle shortest path strategy. Figure 6.25 depicts selected cost maps from the validation set. Objects in the scene develop a more diffuse cost in their vicinity with increasing ϵ. The baseline cost map trained with normal A* (Figure 6.26a), has relatively sharp edges compared to when $\epsilon = 0.05$ in Figure 6.26c.

We have not pursued this strategy further because Field D* (Section 5.1) produces a gradient that, while sparse, is also continuous and not quantized. Combined with a large number of path examples, cost map smoothness is not a problem. However, it is likely that human-labeled datasets



(a) $\epsilon = 0$

(b) $\epsilon = 0.01$

(c) $\epsilon = 0.05$

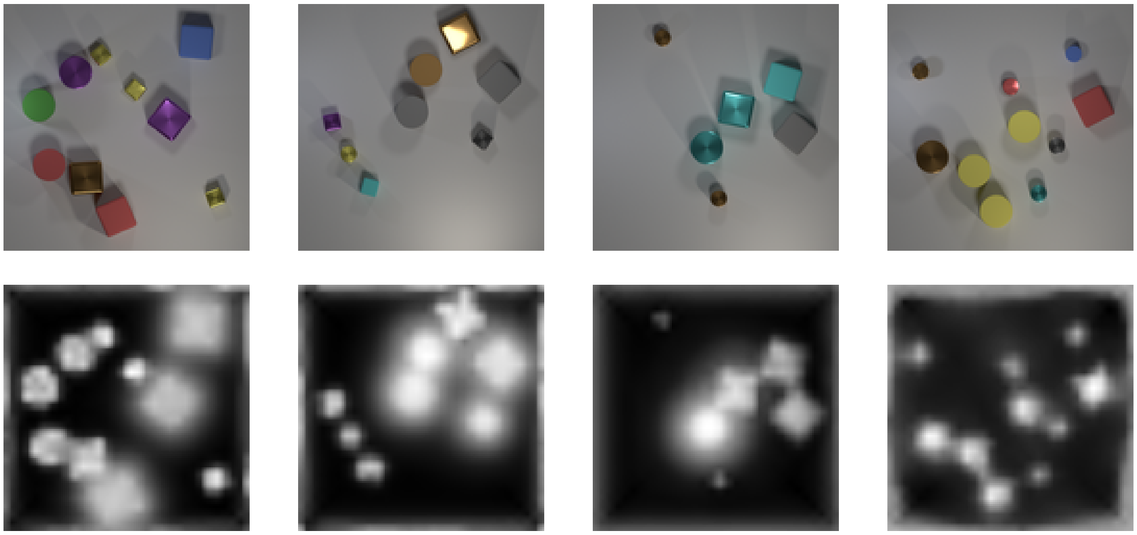
(d) $\epsilon = 0.10$

Figure 6.25: Performance for Epsilon-A* for single runs. $N = 100$, $\eta = 10^{-3}$, $\lambda = 10^{-5}$.

will have fewer training examples per image, again leading to policies that only visit a minority of the map cells during training. Supplementing training with a few sub-optimal states may prove beneficial, with care taken to be sure that there is no loss of performance, as in Table 6.9.

6.6 Semi-Synthetic Dataset and Results

The semi-synthetic dataset combines human-annotated aerial images and synthetically generated commands and trajectories. It is a low-cost stepping stone between the CLEVR-derived dataset and human-labeled navigation datasets, which do not exist for this task to the best of our knowledge.



(a) $\epsilon = 0$ (b) $\epsilon = 0.01$ (c) $\epsilon = 0.05$ (d) $\epsilon = 0.10$

Figure 6.26: Results for Epsilon-A* for various ϵ. $N = 100$, $\eta = 10^{-3}$, $\lambda = 10^{-5}$.

A dataset for this task must be taken from an aerial viewpoint and have pixel or polygonal object labeling that accurately follows object contours, with several objects per scene and multiple object classes. Objects should have some meta-data beyond labels, such as orientation, color, etc. Finally, the dataset needs to be large enough to support training a deep network.

We use the DOTA V1.5 dataset (Dataset of Object deTection in Aerial images, Ding *et al.* [27], [28] and Xia *et al.* [29]) as the basis of our hybrid dataset. It contains more than 2800 images at various scales from different image sources distributed between train (50%), test (33%), and validation sets. The images have varying spatial resolutions centered at about 30 centimeters per pixel and are generally a few megapixels in size. The images are of urban and suburban settings and contain scenes of marinas, ports, airports, and logistics hubs which may support many interesting planning scenarios while not overly cluttered.

Table 6.10: DOTA object categories. Those highlighted in green required in each sample. Those highlighted in red excluded from samples.

Vehicle	airplane	8072	ship	32973	large vehicle	22218
	small vehicle	126501	helicopter	635		
Infrastructure	storage tank	5346	bridge	2075	harbor	6016
	roundabout	437	crane	142		
Sport	baseball field	412	basketball court	529	tennis court	2425
	running track	331	soccer field	338	swimming pool	2181

The objects in each image are individually labeled using bounding boxes with 16 categories (Table 6.10). Vehicles are tagged with an oriented bounding box, which allows us to extend our work towards commands that refer to object orientation and position. The images are not geo-registered, so it's impossible to supplement them with road or terrain data.

The iSAID dataset (Zamir *et al.* [30]) supplements DOTA with pixel-level annotations. The pixel-level instance masks allow us to reconstruct a scene graph that we can export to our synthetic dataset generation script. We describe the process of hybrid dataset generation in Section 6.6.1.

We have considered partially annotated aerial datasets, such as the INRIA Aerial Image Labeling Benchmark (Maggiori *et al.* [145]), ATLAS GOSHEP [146], and a dataset constructed by Yuan *et al.* [147]. These were either too small or sparse, annotated with too few object classes to allow interesting scenarios. Imagery from earth observation satellites, such as the dataset used by Iglovikov *et al.* [148], primarily labels land usage at a granularity too coarse for our work. We have experimented with supplementing the object classes with OpenStreetMap [149], provided geo-referenced images are available, but without satisfactory results.

We have also examined well-developed computer vision datasets, such as ImageNet (Russakovsky *et al.* [132]) and VisualGenome (Krishna *et al.* [68]). While these datasets are expansive with many object classes, the images usually only contain a few objects labeled with bounding boxes and have significant background clutter due to the image perspective and environment.

6.6.1 Hybrid Dataset Generation

The dataset generation process reconstructs a scene graph and uses the iSAID scene mask to identify objects' position, class, orientation, and extent. We use the same scripts for the synthetic dataset with appropriate modifications to the command templates to create a new hybrid dataset. We have adapted the command template for the difference in the scene. We generate 1024 scenes for each of the train, test, and validation splits.

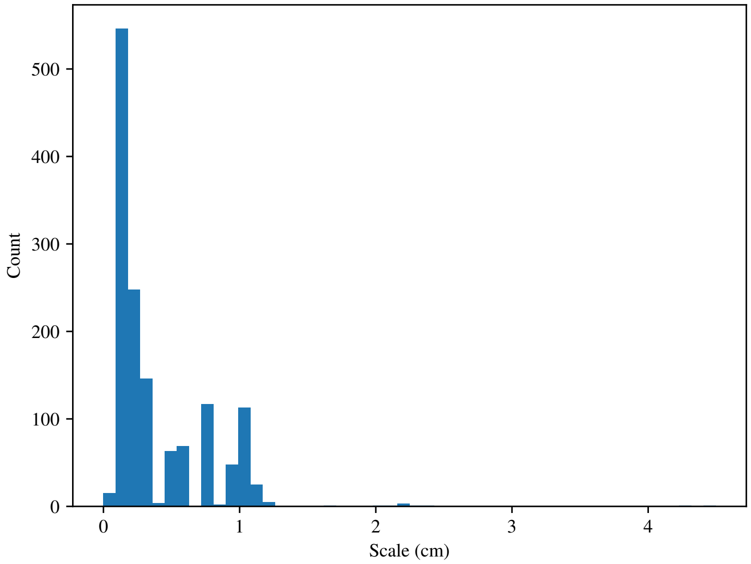
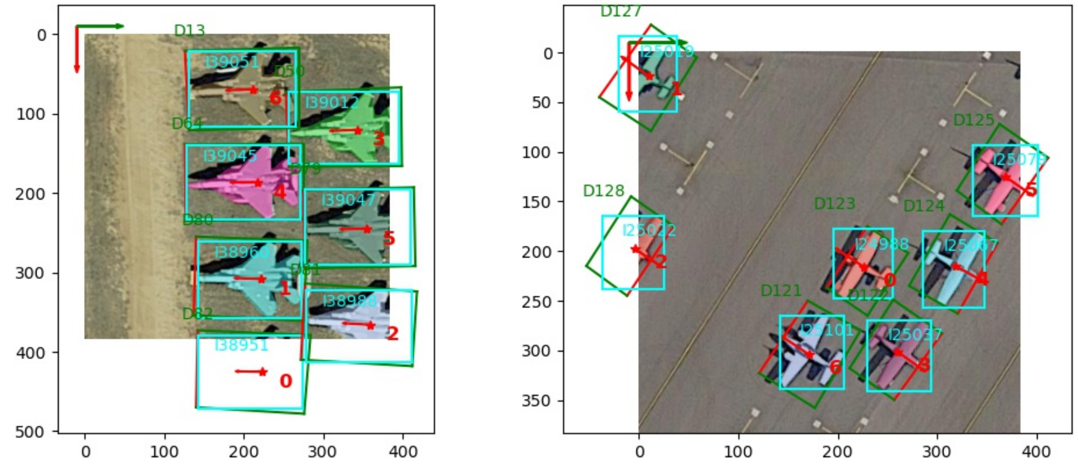


Figure 6.27: Histogram of image scales for DOTA training set. Images with a spatial resolution of less than 0.1 cm per pixel and greater than 0.30 cm per pixel were excluded from our hybrid dataset.



(a) Sampled image

(b) Sampled image

Figure 6.28: Sample images extracted from megapixel source images. We recover object orientation from DOTA and the center of mass and object extent from iSAID, which are combined to create a scene graph indexed by the object identification number and associated object mask (blended colors).

We exclude a fraction of the available images due to substantially different scales or poor image quality (i.e., gray-scale). One of the design goals for DOTA is to test object detection for varying scales from different sources. We exclude images with a scale of less then 0.1cm and greater than 0.30cm (Figure 6.27). We randomly sub-sample the megapixel images into 384 × 384 sub-images. A valid sample must contain an object with a center of mass in a radius that includes 50 percent of the pixels. The object must also fit into the image. In addition, we require the sample to contain at least three objects in total so that we can devise interesting scenarios. Currently, we only include samples with the specific object classes and exclude other scenes with any instances of others (Table 6.10). This was done to simplify the initial dataset and remove expansive objects that overwhelm the free space in the image. Future work may include the sports categories by augmenting the dataset generation script to recognize these as traversable surfaces.

The command set used in this experiment is not aware of object orientation (Table 6.11), so we can augment the data by transforming the scene with small offsets and flipping without changing the meaning of a command. This will not be possible with more elaborate commands that refer to left, right, front, back, etc. The allowable value of the affordance tag, ⟨A⟩, are avoid and trust. The shape tag, ⟨S⟩, can be any of the object categories in the dataset. We generate four instances for each command family to produce a dataset with 1024 scenes and approximately 16,000 training examples for each of the train, test, and validation splits.

Table 6.11: Command types in hybrid dataset. Optional properties in parentheses. Alternative templates (not shown) and word substitution generate a wider variety of commands than apparent in this sample.

Family	Command Template
0	**Single set of objects** Go ⟨A⟩ing [the] ⟨S⟩
1	**Intersection of two sets of objects** Go ⟨A⟩ing [the] ⟨S1⟩(along with) [the] ⟨S2⟩
2	**Union of three sets of objects** Go ⟨A1⟩ing [the] ⟨S1⟩and ⟨A2⟩ing [the] ⟨S2⟩
3	**Subset of a set** Go ⟨A⟩ing [the] ⟨S1⟩(except) [the] ⟨S2⟩

6.6.2 Experiment

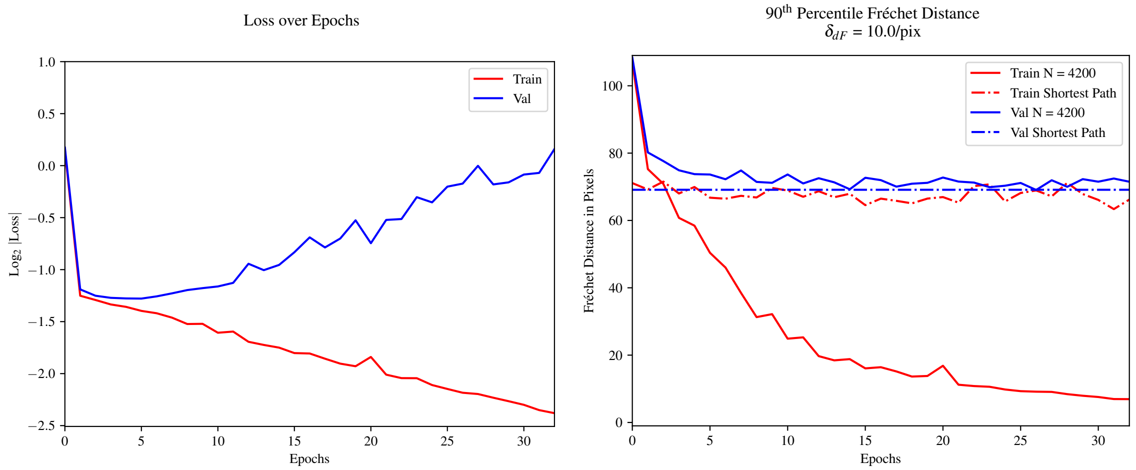
Figure 6.29 shows that the algorithm overfits the training data. Both train and validation splits decrease in loss until the validation loss begins increasing (Figure 6.29a). Figure 6.29b shows that the training Fréchet distance compared to the ground truth path (${}^{90}F_{GT}$) continues to decrease while the validation data does no better than the baseline non-obstacle shortest path (${}^{90}F_{SP}$). This problem affects all command families (Table 6.12). The mean Fréchet distance ($\bar{F}_{GT}$) is significantly larger than the median, indicating that there is a large minority of examples that skew the metrics.

Table 6.12: 90th percentile Fréchet distance in pixels (${}^{90}F_{GT}$) for each command family compared to ground truth (lower is better) and shortest non-obstacle path baseline (${}^{90}F_{SP}$) compared to ground truth. ${}^{50}F_{SP}$ is 13.41 for the training set and 13.56 for the validation set.

	Train					Validation				
Family	${}^{90}F_{SP}$	${}^{90}F_{GT}$	${}^{50}F_{GT}$	$\bar{F}_{GT}$	$\sigma_{F_{GT}}$	${}^{90}F_{SP}$	${}^{50}F_{GT}$	${}^{50}F_{GT}$	$\bar{F}_{GT}$	$\sigma_{F_{GT}}$
All	66.20	6.93	3.21	4.22	5.17	69.11	71.50	14.89	28.14	32.83
0		5.85	3.02	3.64	3.34		65.56	13.92	25.78	29.74
1		5.97	3.00	3.54	2.20		78.48	13.83	28.79	34.80
2		6.67	3.25	4.12	5.17		74.47	15.45	29.11	34.50
3		10.92	4.13	6.21	8.73		74.97	17.57	29.49	31.85

Figure 6.30 shows two images sampled from the training dataset. They are among the worst 16 performers from 4096 randomly selected training examples. The cost maps generated in both examples successfully ignore clutter (unlabeled regions, such as the buildings on the left side of Figure 6.30a) to produce reasonable paths. However, the cost map is less distinct than the synthetic dataset.

In particular, the trailers in Figure 6.30b show aliasing, suggesting that the cost map upsampling technique is reaching its technical limits. In this case, the diagonal orientation of the trailers is misaligned with the axes of the feature map. A fuzzy cost map makes it more difficult for the



(a) Loss curve, $\eta = 10^{-4}$, $\lambda = 10^{-5}$, $N = 150$

(b) 90$^{\text{th}}$ percentile Fréchet loss

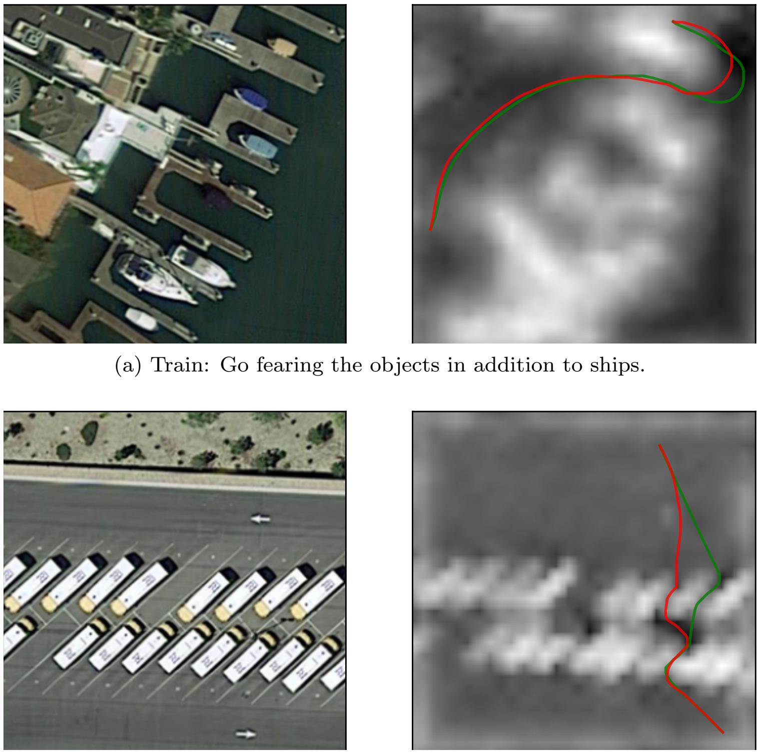
Figure 6.29: Training performance plots for MMP applied to modified DOTA/iSAID dataset. Loss curve in Figure 6.29a shows over-fitting. Baselines in Figure 6.29b (dashed) are Fréchet distance between non-obstacle shortest paths and ground truth paths.

planner to find routes through tight spaces. More results on the training and validation datasets are in Section 8.6.

We have tried several experiments to train a network that generalizes. The work described in Section 6.4 was motivated by this problem. We have made learning easier by augmenting the image data with the object mask and simplifying commands to one word: "Go." None of these were successful, which leads us to believe there is a problem with path learning.

To investigate this, we modify our CLEVR-derived dataset with fewer examples per scene and get similar results. Table 6.13 show that as the number of examples per command family per scene becomes smaller than 8, the algorithm has trouble generalizing. We believe that the problem may result from insufficient examples for the algorithm to learn to generalize, given that each example path visits just a fraction of the map cells.

This trend is reflected in the Fréchet loss curves in Figure 6.31. A dataset with just 1 example of each command family per scene will have 32 times fewer examples per epoch than the largest dataset. It is important to keep this in mind when comparing the plots. However, it is clear that with only 1 example per command family, Figure 6.31d shows a different shape than the other plots. The training error has reduced significantly with no sign of the validation loss beginning to reduce. It is plausible that the DOTA/iSAID-derived dataset is too sparse to train properly. This may be resolved by increasing the number of training examples, which is trivial for a synthetic dataset. However, realistic human-labeled datasets will not be so rich in examples. Efficient learning from limited samples may be an area of future research.



(a) Train: Go fearing the objects in addition to ships.

(b) Train: Trust the things and trust the large vehicles.

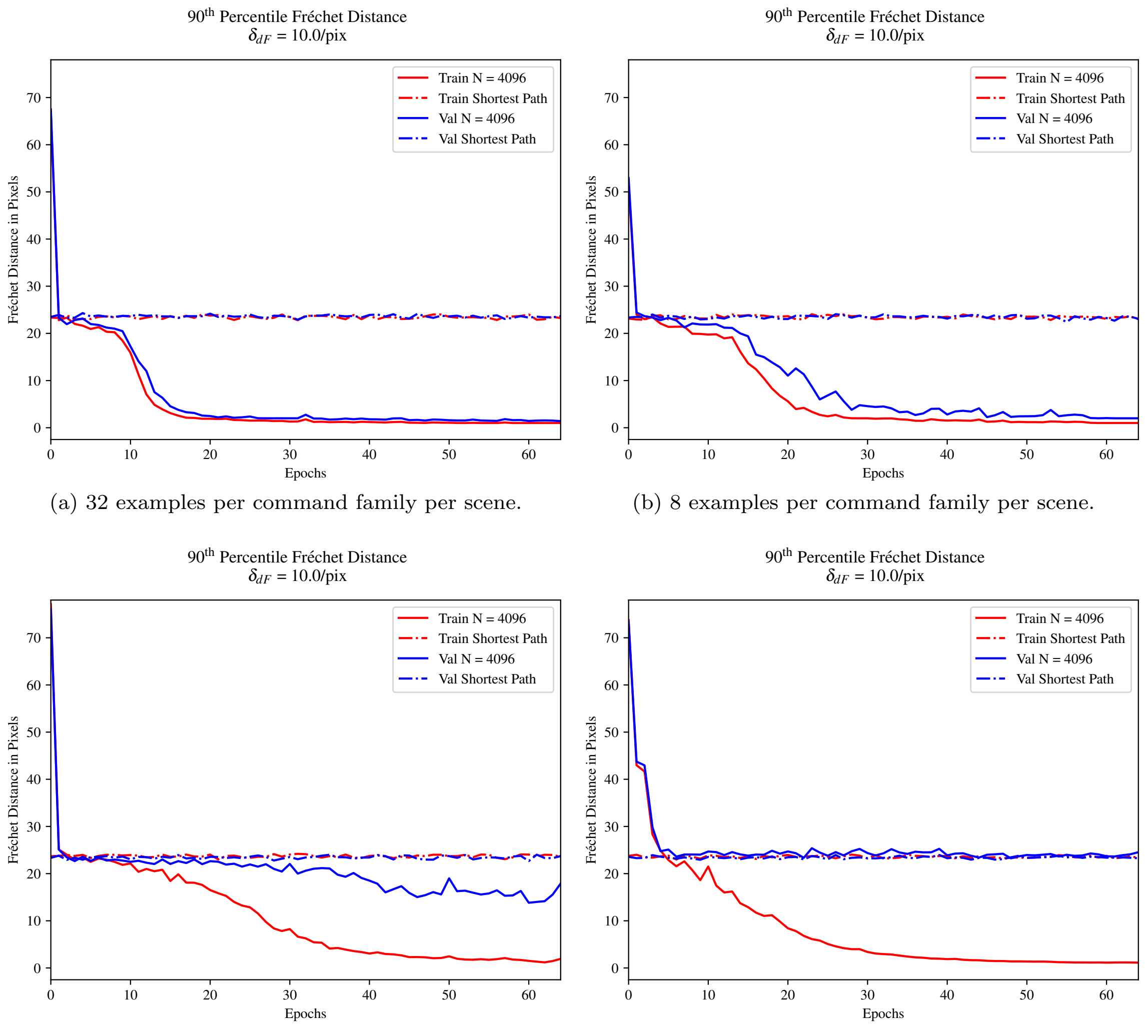
Figure 6.30: Worst training examples from sample of 4096 for DOTA/iSAID experiment. Source image (left), generated cost map (center), error gradient (right), ground truth path (green), generated path (red).

Table 6.13: Training and validation performance suffers when the number of examples per family per scene (K) are reduced. Reported values are ${}^{90}F_{GT}$ in pixels using CLEVR-derived dataset.

		Command Family											
		Training						Validation					
		0	1	2	3	5	All	0	1	2	3	5	All
K	1	2.48	3.39	3.24	2.98	2.36	2.94	21.77	28.32	27.43	27.38	23.00	24.31
	4	4.15	6.48	6.32	11.88	4.10	6.28	15.32	22.77	22.02	22.38	16.54	20.62
	8	1.17	3.43	2.00	2.00	1.34	1.96	1.66	14.17	10.50	4.75	2.00	4.51
	32	1.05	5.70	2.22	1.52	1.22	1.77	1.33	12.45	4.85	2.16	1.55	2.75

6.7 Summary

In this chapter, we have demonstrated the deep planning algorithm using a synthetic dataset. Removing language as an input shows that it uses the command to modulate image features to generate a cost map. This supports the concept that a deep network can produce trajectories using an image and a command without an explicit model of the latent process that generated the ground



(a) 32 examples per command family per scene.

(b) 8 examples per command family per scene.

(c) 4 examples per command family per scene.

(d) 1 example per command family per scene.

Figure 6.31: Performance plots for a deep network with MMP with fewer examples per command family per scene. $\eta = 10^{-4}$, $\lambda = 10^{-5}$, $N = 256$.

truth. There is a clear benefit to biasing minibatch selection towards more informative examples. Importance sampling techniques may also be an excellent way to identify valuable examples without making strong assumptions about the underlying model.

However, this work is not complete and not without limitations. We have yet to show better than baseline performance with more realistic data. Improving this result is necessary before testing with human-labeled data and practical application. The number of examples per image or the information content of each trajectory may be insufficient for the algorithm to generalize. Our synthetic and hybrid datasets, and the revised CLEVR tools to generate them, are a valuable contribution to researchers developing planning and navigation algorithms that operate in a similar context. To the best of our knowledge, there are no similar datasets currently available.

The loss function effectively uses a generalization of the Hausdorff metric as a loss function instead of the Fréchet coupling, which is more appropriate for paths. While the Hausdorff metric does approximate the Fréchet metric for small errors, this may direct the optimization towards local minima early in training when this approximation is less appropriate and may be especially problematic with noisier, naturalistic image data. Future investigators may wish to experiment with alternative loss functions within the MMP framework.

Chapter 7

Conclusions & Future Work

In Chapter 1, we establish that there is a need for a field robot navigation system that combines language understanding, perception, and path planning with a single loss metric. As a surrogate to this very complex problem, we study the simplified problem of a notional robot that must navigate through a static aerial scene, reducing complexity without losing the fundamental challenge of this work.

In Chapter 2, we explore relevant research to understand how others have approached similar problems so that we may improve upon and contribute to the state-of-the-art. Most existing research targets structured environments for which topological maps are best suited, such as indoor environments, where the agent may take a few discrete actions at a relatively small number of junctions. Techniques based on reinforcement learning are suitable for this kind of problem.

In contrast, our notional robot operates on a metric graph, often with no identifiable decision points. Without knowing the relative value of visiting one state over another, we believe that inverse reinforcement learning techniques are suitable. Max Margin Planning and Maximum Entropy IRL are two solutions to the ill-posed problem of reconstructing the latent value function of an MDP given a limited number of input-output observations. We initially selected Max Margin Planning as the vehicle for our research because it is less numerically demanding on the CPU, as it uses efficient planners in A*/Field D* to find optimal paths through a cost map. However, our later move towards the Bellman-Ford algorithm as part of an all-GPU implementation makes MaxEnt IRL an appealing alternative.

Without prior work that precisely matches the task, we study network architectures from the related field of Visual Question Answering, using attention maps as an analogy for the cost maps used in planning. We select the FiLM architecture because it makes few assumptions about the underlying process used to generate the training and testing data, unlike its contemporary networks. This model is appropriate since the model humans use to perform similar tasks is often inscrutable.

In Chapter 4, we develop the required techniques to make MMP a stable loss function suitable for training a deep network over millions of iterations. We show in Chapter 5 how MMP can use the Field D* algorithm to produce more naturalistic paths. In Chapter 6, we combine FiLM with MMP and a newly developed synthetic dataset to explore how the algorithm performs under controlled conditions and present several analyses. We also experiment with a hybrid dataset that uses human-labeled aerial imagery with computer-generated commands and paths as the first step towards an entirely human-annotated dataset.

Finally, in Section 5.3, we show how the algorithms used in this thesis translate to highly parallel GPU architectures to facilitate accelerated learning. Over the past decade, the rapid progress of deep learning has been driven by libraries of modular and highly optimized parallel algorithms provided by frameworks such as TensorFlow, PyTorch, and others. As similar planning elements did not exist for our task, we have built our own, deliberately selected algorithms that efficiently use the vast computational power of contemporary GPUs.

7.1 Key Findings

We have shown that it is possible to fuse perception, language understanding, symbol grounding, reasoning, and planning into a single differentiable process for generating trajectories on a map. These elements are discrete modules that are designed and trained separately in prior navigation systems. There are two primary benefits to integrating the modules into a single system.

First, difficult to define human concepts in language, perception, and planning are captured by directly imitating expert behavior. Our goal is to develop a navigation system for a hypothetical robot that can operate in a team with humans. This requires that it communicates naturally, perceives the world as humans do, and acts in a manner consistent with other humans. Without the ability to directly model this latent process, we must instead learn by example. Although we have shown results only with synthetically generated data, we feel that the approach is the correct one.

Second, modular systems are often trained on sub-tasks that may not directly support the overall task metric. There is no feedback from the task-level goal to adapt the parts to perform optimally as a system. For example, a perception module may treat all object classes with equal importance when some are not required for the task. These modules and their interfaces have been removed from the system in favor of an end-to-end, data-driven approach.

VQA systems internally perform many of the tasks required of navigation systems, primarily symbol grounding, logical inference, and attention mapping. The FiLM architecture is one of many VQA models that may adapt to form the core of a navigation system simply by changing the output function to produce cost maps instead of classifications. We have shown that our architecture learns and generalizes when trained with synthetic data. Language inputs improve performance beyond baselines, demonstrating that the algorithm and dataset require language and perception to achieve good performance.

We have adapted Max-Margin Planning as our planning loss and have shown that it effectively trains a deep network to generate cost maps by mapping commands and image features to trajectories. In a novel combination with Field D*, it can produce smooth human-like trajectories, which are needed when learning from human-labeled data. In contrast, prior work produces linear paths because it uses A* in combination with MMP.

Planning on the CPU is a computational bottleneck that prevents the algorithm from utilizing the GPU effectively. We have modified Field D* to use the Bellman-Ford-Moore algorithm, which is more scalable and time-efficient on the GPU, but less work-efficient than our initial implementation on the CPU. With the CPU no longer limiting performance, we note that the Bellman-Ford algorithm has similar runtime to Value Iteration and presents an opportunity to explore MaxEnt IRL as an alternative loss function.

In summary, our contributions are:

- A network architecture for unifying natural language processing, image understanding, symbol grounding, and path planning into a single deep learning framework with a single objective function. We evaluate the algorithm on two datasets, highlighting strengths and weaknesses and identifying directions for future work.
- An extensible synthetic dataset for testing the reasoning abilities of this and other similar systems in an unbiased environment with virtually unlimited data, based on the CLEVR dataset (Johnson *et al.* [25]) and DOTA/iSAID dataset (Ding *et al.* [28] and Zamir *et al.* [30]), but re-targeted for our problem space.
- A scalable, massively parallel implementation of the Maximum Margin Planning, Field D*, Fréchet and Modified Hausdorff coupling metrics that enable efficient training and evaluation of deep networks with planning losses. These tools are an essential bridge between a theoretical contribution and a practical method.

7.2 Basic Extensions

This thesis demonstrates the viability of fusing language processing, scene understanding, and planning into a deep network with a single loss function as a future navigation system for field robots. Nonetheless, there remain many required enhancements to make the work practically useful.

Benchmark Against Alternative Navigation Systems

We have not performed a baseline comparison against other navigation systems, such as Oh *et al.* [11]. Robot navigation systems are complex and often highly specialized to a particular task and sensor set. It isn't easy to find a common dataset for a fair comparison between systems, and this is further complicated by the need for large datasets to train deep learning models.

While difficult to implement, this comparison would be very insightful. For example, Oh *et al.* [11] use a command language backed by a grammar with a precise meaning, unlike our system, which infers the meaning of commands from example path data without any formalism. One can expect that some commands, perhaps compound statements or recurrent constructions, will cause errors for one method and not another. This analysis may uncover the kinds of commands that elicit erroneous plans and evaluate how this affects the utility of a robot.

Survey of Contemporary VQA

We studied a few VQA network architectures in section 3.3, evaluating the suitability of architectures that were state-of-the-art at the commencement of this work. Much has changed in the interim. While conceptually straightforward, modifying the base network in our model requires some adaptation and innovation. When the loss function and dataset are more mature, a comparison of different network architectures may give insight into the kinds of commands and concepts that a given architecture has the capacity to learn.

The use of language as a sensor is a concept that features prominently in Duvallet [52]. Commands and the information they convey are intimately tied to the scenes they describe. Taken to an extreme, a command such as: "Go to the cube hidden behind the sphere" requires the algorithm to find the sphere and speculate on the features of a hidden object through language. Deep networks with fine-grained, joint embedding are one way to realize this concept without requiring an explicit process to integrate the two sources of information into a single world model. Newer VQA architectures may use better methods for joint embedding of language and image features than FiLM.

Learning from Pixels to Plans

We use pre-trained mid-level ResNet features (He *et al.* [82]) as the input to our algorithm. That a model trained for a different purpose on a vastly different dataset is useful for our task speaks to the notion that deep networks learn fundamental concepts in vision. We use pre-trained features to reduce the number of free variables in our model to make it more tractable, both computationally and theoretically. True end-to-end learning can be achieved by fine-tuning a pre-trained model and adapting it to the task and data.

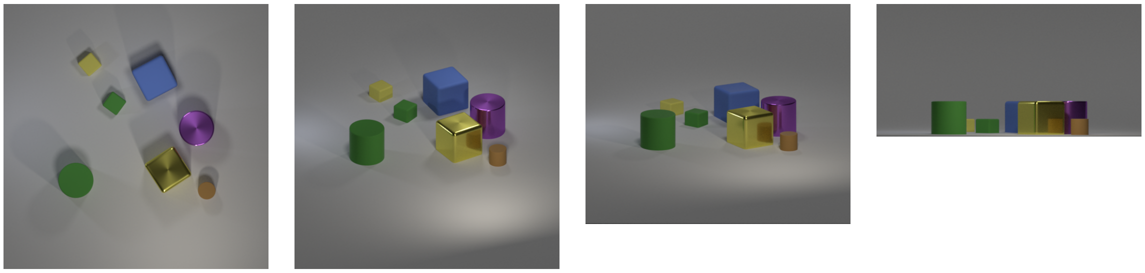
We use ResNet features because that is what FiLM uses. However, our model is utterly agnostic towards the number of channels in the feature map and mildly affected by changes in the map dimensions. Therefore, different stem networks may be substituted for ResNet with minor work. Some stem networks are likely better suited to our task than others.

Maximum Margin Planning with Fréchet Metric

The L_2 version of the Maximum Margin Planning loss effectively minimizes the cost of a path over a cost map augmented with a form of Hausdorff coupling metric. The Fréchet coupling metric more accurately represents the conformity between paths, whereas the Hausdorff metric is better suited to measuring the conformity of point clouds. When the paths are substantially non-conforming, the loss does not represent the true quality of the path. Early in training, when the Hausdorff and Fréchet metrics are likely to be very different, the L_2 loss gradient will direct the model towards weights which increase the conformity between *point clouds*, but may not reflect the information required to produce more congruent *paths*.

Max Margin Planning intentionally discards information crucial to the problem by using the Hausdorff metric as a proxy to the Fréchet metric. At the same time, MMP is computationally efficient precisely because the minimizer at each step, μ^*, is recovered using a dynamic program such as Field D* or Dijkstra's algorithm. The simultaneous optimization of the path and the Hausdorff metric is computed by adding a distance transform of the exemplar path, $\ell(x_i)$ to the objective (Equation (4.2)).

Conversely, there is no apparent efficient solution to simultaneously optimize a path through a cost map with an additional Fréchet coupling metric objective. The planner and Fréchet coupling



(a) 90 degrees (b) 75 degrees (c) 60 degrees (d) 0 degrees

Figure 7.1: CLEVR scene rendered at varying angles to test planning with occlusion.

metric have efficient solutions in the same feasible space but operate in different state spaces (paths vs. path couplings). An approximate solution may yield results comparable to an exact one without undue complexity. Alternatively, a computationally expensive yet exact solution may be used in the initial training phases, when the Fréchet metric significantly differs from the Hausdorff metric, to direct the model towards a local minimum more suitable for learning paths, as opposed to point clouds.

Spatial Reasoning with Perspective Images

The aerial images used in this work are a necessary simplification to develop a system without the complication of hidden objects. Robots in the real world do not have this level of omniscience, and most do not have the benefit of aerial imagery. A system that views the world at an oblique angle must plan a path while reasoning about hidden and partially obstructed objects and the space they occupy. For example, the command ”Go to the green cube behind the purple cylinder” must generate a reasonable plan even if the green cube is entirely obstructed, based on the knowledge that it is somewhere behind the purple cylinder. This knowledge may come from prior experience, as encoded in the network weights, or from the use of language as a sensor, as developed by Duvallet [52].

The CLEVR dataset generation scripts can synthesize images from varying angles (Figure 7.1), all with the same command and ground truth paths. Ideally, the cost map generated for an image with 60-degree obliquity should be the same as one generated with a 90-degree obliquity. The reality is that this transformation loses some information. In addition to occluded objects, some degree of three-dimensional information must be recovered from a two-dimensional projection of the scene.

For the case of Figure 7.1b, this transformation may be approximated with a homography. In more extreme cases, such as Figure 7.1d, a reconstructed cost map would require information about the relative sizes of objects to disambiguate scale and distance, spatial reasoning to determine which objects are occluded by others, and other prior knowledge and assumptions to generate a reasonable cost map with incomplete information.

Planning with perspective transformations and occlusion using differentiable transforms within deep networks has been demonstrated by Gupta *et al.* [36]. While we do not know if our architecture can effectively learn to plan under similar conditions, a synthetic dataset allows us to study candidate architectures in a controlled fashion. In particular, a fully synthetic benchmark will enable us to learn how the generated cost map in the occluded regions differs from the ideal cost map, similar to the study in Section 6.2.

Endpoint Prediction and Multi-Segment Paths

Our current architecture accepts commands that describe *how* to get from one place to another but can not determine *where* to go. Another limitation of the planner is that it cannot produce self-crossing paths. This precludes commands such as: "Go around the car." It is a side effect of the dynamic programming techniques common to planners like Dijkstra's algorithm. To mitigate this, we envision a network architecture with a second output channel encoding a probability distribution of likely destinations and a cost map, trained with a joint objective.

Max Margin Planning is an efficient discriminative method for structured prediction. It predicts a single solution for a given set of inputs, lacking the flexibility of MaxEnt IRL, which generates predictions from a distribution. Our approach would create a hybrid system, merging a high-dimensional discriminative classifier for path generation with a low-dimensional generative classifier to predict endpoints, lending flexibility to the system. This architecture will require a loss objective that combines the planning loss with an endpoint loss in the form of a cross- or max-entropy loss.

This architecture may use a recurrent network to generate intermediate waypoints to break a path into segments individually expressible by the planner. This will allow the system to solve commands with waypoint path constraints such as: "Go to the front of the house and then to the car." This recurrent network also introduces a more granular level of planning for intermediate goals, with MMP as the local planning algorithm. Multi-resolution planning is a common feature in practical robot navigation systems that must efficiently plan over larger scales.

Position and Orientation Encoding

We have experimented with encoding the start position of the agent by augmenting the ResNet feature map with additional channels. We have experimented with various methods to embed the information but are not satisfied with the results and feel that more work is required. While the commands that rely on this information perform better than baseline (Table 6.3), they also perform worse than other command types. In addition, the network must encode the agent's orientation to support future work with commands that reference the agent's point of view.

FiLM already augments the feature map with two channels, encoding the feature's x and y coordinates. We follow their lead and embed an additional channel encoding the start position, but there are other places in the network where this information may be helpful. We have experimented with embedding the start location in the initial hidden state of the language encoder to influence the cost map generation process through the FiLM modulation vectors. While preliminary experiments did not seem promising, it is reasonable to assume that *both* the interpretation of the command and the image features should be influenced by the position and orientation of the agent. A more thorough investigation on how to embed this information is required.

Expanded Synthetic Dataset

The CLEVR- and DOTA-derived datasets should be expanded to test more elaborate scenarios than those presented in this thesis. An expansive dataset will reveal the limits of this architecture

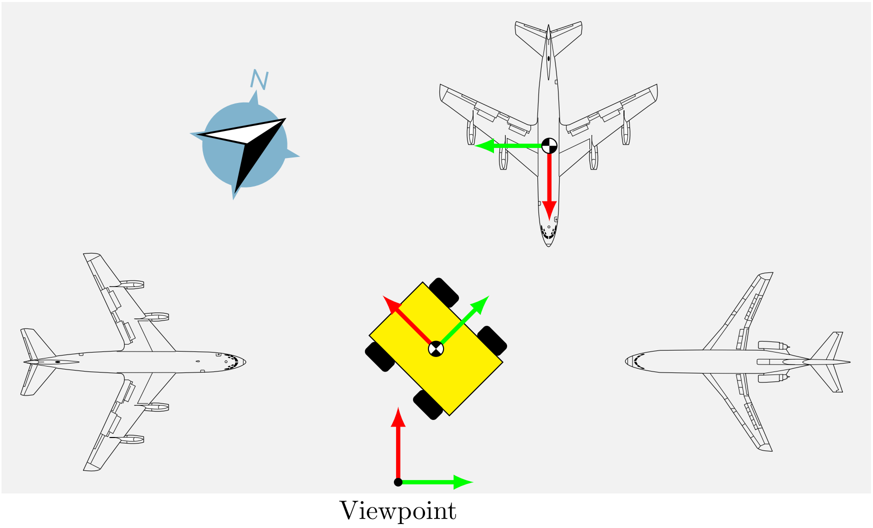


Figure 7.2: Viewpoints for navigation commands.

and may point towards networks or loss functions better suited for navigation than the one we have chosen. We expect there are some concepts that our model simply can not learn.

In the CLEVR-derived dataset, commands are given with reference to the observer. The position and orientation of the observer ("Viewpoint" in Figure 7.2) are implicit in the image. This viewpoint defines the relative position and orientation of objects to each other. Further, the objects in CLEVR do not have a discernible orientation.

However, there are additional viewpoints that we have not yet considered, especially when objects have unambiguous orientations. Figure 7.2 is a scenario within which a command may refer to objects relative to the absolute coordinates (compass), the robot, and other objects. Expanding the dataset to include these kinds of commands is a necessary step towards a more evolved navigation system. It must coincide with some of the architectural developments mentioned above.

Extending the dataset is not technically challenging because a modular scripting language generates plans. Table 6.13 includes a promising result for command family 5, which is defined in Table 6.1. This command family tests the model's ability to identify objects that are nearest or farthest from the agent's starting position. Table 7.1 contains some of the templates that we plan to use in an evolved navigational system.

Ultimately, the system should be tested against data entirely generated by humans: human commands and human-generated paths with natural images. Human-labeled data is expensive and time-consuming to collect, and this should not occur until the algorithm has demonstrated good performance on a large corpus of synthetic scenarios. The network and loss function must evolve concurrently to ensure that the system can imitate human proficiency in translating commands and images into paths.

Human-Labeled Dataset

A human-labeled dataset is essential for testing a system designed to interface a robotic teammate with its human companions. Our synthetic dataset contains commands and paths that have all been generated using a single latent process. In contrast, there is significant variation in the thinking processes of individual humans. Human-labeled data introduces a new level of complexity and variability not captured by the synthetic dataset.

Collecting and annotating enough data to train a deep network is a daunting and expensive task that we have delayed until the model is mature enough to justify the investment. Still, it is a necessary step towards developing a practical navigation system.

7.3 Towards a Practical Navigation System

We have identified a set of qualities in the literature that practical navigation systems for a ground robot must possess. Our model requires significant architectural changes to address these points.

1. Produce stable paths over time.
2. Accumulate information about the world that has been observed.
3. Operate from a ground-vehicle perspective in a world that is never fully observable.
4. Efficiently explore when there is no definitive plan.
5. Reason about topological and metric command constraints.
6. Declare success or detect a failure and ambiguous situations.
7. Integrate additional sources of information and demonstrate compatibility with existing algorithms.

Time is the most critical element of a practical navigation system *not* addressed by our architecture. Our model assumes the world is static and that all information is available to the system at once. Time introduces a new planning horizon, beyond which there is information that requires the robot to re-evaluate and re-plan to maintain optimal behavior. This planning problem is similar to reasoning beyond an obstruction, as introduced in Section 7.2, but with much more profound consequences to the system architecture. Note that we continue to assume the world is static, just not fully observable at $t = 0$.

Algorithm 20 is a sketch of a simple planning loop that addresses some of the qualities enumerated above. The navigation systems we have surveyed (Oh *et al.* [1] and Hemachandra *et al.* [35], and others) are similarly invoked in a loop without information about previous plans that can affect future decisions.

While each contains a map that is updated periodically, the grounding and planning operations are independent of prior plans. The planning loop will yield the same result at each iteration given a static and fully observable world. However, new information accumulated on the map may make another plan more desirable in a partially observable world, perhaps leading to radical and undesirable changes. During experiments detailed in Oh *et al.* [1], we observed robot behavior that may be explained in this way.

Another problem is that the map update (Algorithm 20 Line 7) is independent of the planner.

Algorithm 20 Simple planning loop with map memory. Λ is command, $\mathcal{I}$ are image/sensor inputs, M is map store, X is robot state.

```
function PLANNINGLOOP($\Lambda$, $\mathcal{I}$, $M$, $X$)
    success ← False
    failure ← False
    $t \leftarrow 1$
    $M_0 \leftarrow \emptyset$                                          ▷ Initialize map
    while $\overline{\text{success}}$ and $\overline{\text{failure}}$ do
        $M_t \leftarrow$ UPDATE($M_{t-1}$,$X_t$,$F(\mathcal{I}_t)$)        ▷ Store new information in map
        $p_t \leftarrow$ PLAN($M_t$,$X_t$,$\Lambda$)                    ▷ Update plan
        $X_{t+1} \leftarrow$ EXECUTE($p_t$)                           ▷ Execute one step
        success, failure ← COMMANDCOMPLETE($\Lambda$, $I_t$, $X_{t+1}$, $M_t$)
        $t \leftarrow t + 1$
    end while
    return success, failure
end function
```

This prohibits end-to-end training with the planning loss, and the system can not learn from or store historical information that affects task performance. There is also no apparent strategy to combine prior information with updates.

7.3.1 Integrating Memory, Planning, and Strategy

Zhang *et al.* [150] explore map building with a deep network using concepts from Graves *et al.* [151] for a location addressable memory that is also a differentiable recurrent network. They point out this is a feature not found in other navigation systems that integrate neural networks, such as Gupta *et al.* [36]. The feature and memory updates are part of a single recurrent network, solving several problems outlined above.

Gupta *et al.* [36] demonstrate that a deep network can transform sensor information from the robot's perspective into an aerial map that conveys the potential reward for moving into a map cell. This model is trained with a loss function that operates in the aerial frame and does not require an explicit model for feature transformation. Gupta *et al.* [36] also produce a confidence map that may be useful for planning with uncertainty, which integrated robot systems sometimes do not model adequately.

In an integrated system, the planner is no longer independent at each iteration of the planning loop. Therefore it will need to train over a sequence of plans generated as the robot explores as part of some greater strategy. This suggests a two-part objective function (Equation (7.1)). In this equation, the planning loss is the loss function we have studied in this thesis and reflects the quality of the current best path given all evidence. The strategy loss scores prior plans in light of recent evidence, encouraging effective exploration and integration of prior knowledge to predict future events. The two components may be identical, or the strategy loss may also include metrics on how long the robot takes to reach its goal, how much the solution paths change over time, or how often the robot fails to find a solution.

$$L = \underbrace{L_P(p_n)}_{Planning} + \sum_{i=1}^{n} \underbrace{L_s(p_i)}_{Strategy} \tag{7.1}$$

7.3.2 Other Goals

If the world is not fully observable at $t = 0$, then it is possible that some referent in a command is not observable. Existing robot navigation systems use topological maps that parallel metric maps to plan with unseen objectives. The topological map conveys the relationship between objects without requiring specific knowledge about where they exist. This could happen if a command refers to an object that is not in view, but that is suspected to be behind a visible object and might come into view in the future from a different location. Duvallet [52] solve a similar problem by generating phantom goal points in a metric map to provide hypotheses over which their model can reason and plan.

It is unclear how to integrate topological and metric mapping into our framework. However, Gupta *et al.* [36] have demonstrated a deep network that forecasts the likely location of unseen objects in the context of planning. The hypothetical target is not explicitly modeled but results from the deep network's integration of prior knowledge with current inputs.

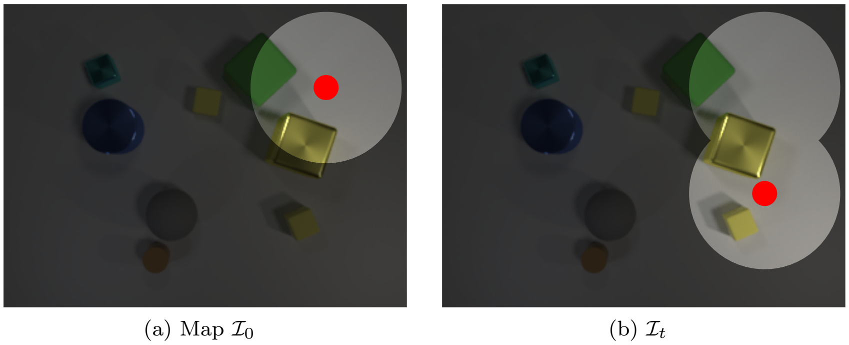
Many problems arise when integrating new technology into an established platform. In particular, robots have many other real and virtual sensors. A robust navigation system naturally integrates many sensor inputs, not just imagery. Combining all these sensing modalities into a single system will be challenging.

The system needs to know when it has succeeded, when it has failed, and when it is lost. Low confidence in task completion should prompt the higher-level intelligence processes to enter a recovery mode or prompt the operator for clarification. This implies a capacity for interaction that is more complex than simply providing a command.

7.3.3 Planning in Simulation

One of the challenges of this work has been to find large amounts of labeled data for our experiments with the right properties. Planning with time will require labeled *sequences* of commands and images, which will be very hard to find and expensive to develop.

A simplified environment may be sufficient to address some of the abovementioned points. We propose to use CLEVR and Blender in the planning loop to simulate time for an agent with a limited field of view (Figure 7.3). However, Blender is not a very good choice for high-speed rendering. Open-source simulation environments, such as AirSim (Shah *et al.* [152]), are well-suited for this task.



(a) Map $\mathcal{I}_0$

(b) $\mathcal{I}_t$

Figure 7.3: A simple experiment using Blender in the planning loop to simulate time. Shaded area is beyond sensor range for agent (red). Map of the world grows as agents moves through scene (Figure 7.3b).

7.4 Conclusion

This thesis has developed one small piece of a much larger robot navigation architecture. New techniques in machine intelligence have allowed us to re-think what this architecture looks like and how the pieces fit together. Deep learning enables us to internalize the many different elements of traditional architectures into a monolithic process. While this often leads to better performance, end-to-end learning is not without challenges.

Deep learning techniques typically require large amounts of labeled data and training time. As tasks become more complex, we believe that simulation environments are one way to solve this problem. We have posited just a few of the qualities an evolved navigation architecture would require to be practical, and a simulated environment can help develop most of them. However, a good teammate must also anticipate human needs and meet human expectations, which are difficult to replicate algorithmically. To be of practical value, a system trained on massive amounts of simulated data must transfer this knowledge to the real world and supplement its experience with data that is unpredictable, imperfect, and precious.

Table 7.1: Proposed viewpoint relative command types for revised synthetic dataset.

Major	Minor	**Direct object identification, simple affordances**
0	0	Go ⟨A⟩ing [the] (⟨Z⟩ ⟨C⟩ ⟨M⟩) ⟨S⟩.
0	1	Go ⟨A⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (along with) [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
0	2	Go ⟨A1⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ and ⟨A2⟩ing [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
0	3	Go ⟨A⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (except) [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
0	4	Go ⟨A1⟩ing [the] (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ or [the] (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
Major	Minor	**Direct object identification, complex affordances**
1	0	Go ⟨A⟩ing [the] ⟨R⟩ side of (⟨Z⟩ ⟨C⟩ ⟨M⟩) ⟨S⟩.
1	1	Go ⟨A⟩ing [the] ⟨R1⟩ side of (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (along with) [the] ⟨R2⟩ side of (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
1	2	Go ⟨A1⟩ing [the] ⟨R1⟩ side of (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ and ⟨A2⟩ing [the] ⟨R2⟩ side of (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
1	3	Go ⟨A⟩ing [the] ⟨R1⟩ side of (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (except) [the] ⟨R2⟩ side of (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
Major	Minor	**Indirect object identification, simple affordances**
2	0	Go ⟨A⟩ing [the] object to the ⟨R⟩ of the (⟨Z⟩ ⟨C⟩ ⟨M⟩) ⟨S⟩.
2	1	Go ⟨A⟩ing [the] object to the ⟨R1⟩ of the (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (along with) [the] object to the ⟨R2⟩ of the (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
2	2	Go ⟨A1⟩ing [the] object to the ⟨R1⟩ of the (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ and ⟨A2⟩ing [the] object to the ⟨R1⟩ of the (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
2	3	Go ⟨A⟩ing [the] object to the ⟨R1⟩ of the (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (except) [the] object to the ⟨R1⟩ of the (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
Major	Minor	**Indirect object identification, complex affordances**
3	0	Go ⟨A⟩ing [the] ⟨R1⟩ side of the object to the ⟨R2⟩ of the (⟨Z⟩ ⟨C⟩ ⟨M⟩) ⟨S⟩.
3	1	Go ⟨A⟩ing [the] ⟨R1⟩ side of the object to the ⟨R2⟩ of the (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (along with) [the] ⟨R3⟩ side of the object to the ⟨R4⟩ of the (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
3	2	Go ⟨A1⟩ing [the] ⟨R1⟩ side of the object to the ⟨R2⟩ of the (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ and ⟨A2⟩ing [the] ⟨R3⟩ side of the object to the ⟨R4⟩ of the (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.
3	3	Go ⟨A⟩ing [the] ⟨R1⟩ side of the object to the ⟨R2⟩ of the (⟨Z1⟩ ⟨C1⟩ ⟨M1⟩) ⟨S1⟩ (except) [the] ⟨R3⟩ side of the object to the ⟨R4⟩ of the (⟨Z2⟩ ⟨C2⟩ ⟨M2⟩) ⟨S2⟩.

Chapter 8

Appendix

This chapter contains notes on the dataset generation process (Section 8.1) and additional examples from error analyses. Section 8.2 shows the worst-case examples for the network in Section 6.1.1. Results for the different error regions for all command families can be found in Section 8.3 and Section 8.4. Finally, Section 8.5 contains additional examples of the effect of language on cost map modulation.

8.1 Synthetic Dataset Generation

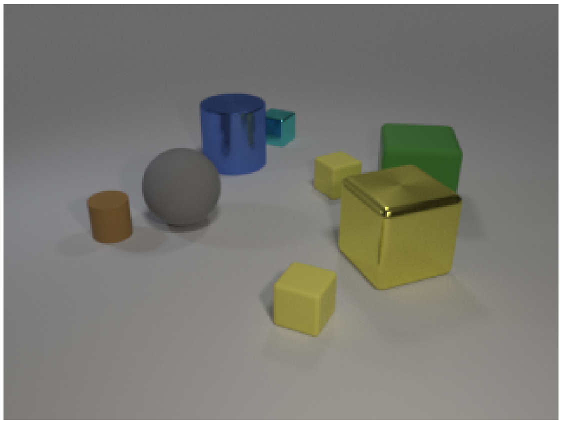
We build upon the work of Johnson *et al.* [25] and extend the CLEVR dataset to include new functions suitable for path generation instead of visual question answering.

Figure 8.1 is an example of a typical CLEVR scene. Each object has one of three shapes, eight colors, two materials, and two sizes. Between three and ten objects are randomly generated and placed in the scene, ensuring that no object significantly obstructs another from the camera's point of view. The data generation process produces scene graphs that contain all information used to create the image. It is then possible to generate questions with known answers and controlled properties algorithmically. For example, queries with trivial answers are usually rejected so that they do not dominate the dataset.

We first modify the image generation script to produce orthonormal images and generate scene masks to identify each object and its extent in the image uniquely. In this dataset, we remove spheres from the list of allowable objects. It was thought they would be indistinct from cylinders when viewed from above. However, the planning algorithm has proven sensitive to the specular shading of metallic surfaces and the diffuse shading of rubber. It may also be sensitive to the unique shading of a spherical object when viewed from above.

With the scene graph and the object mask, the system has complete knowledge of the properties of every object, which allows us to generate synthetic paths using the following steps.

Figure 8.2 is an example JSON command template for command of family 1. The template



```
Object 0
  large blue cylinder made of metal
  3D Coords: -2.42221355 0.742926836 0.699999988
  Pixel Coords: 133 75 13.246380806
  Rotation: (deg) 222.084440533
Object 1
  large gray sphere made of rubber
  3D Coords: -0.901645839 -1.57468926 0.699999988
  Pixel Coords: 103 106 10.8944158554
  Rotation: (deg) 203.260260775
Object 2
  large yellow cube made of metal
  3D Coords: 2.51925992 0.255135595 0.699999988
  Pixel Coords: 227 127 9.6789894104
  Rotation: (deg) 7.52761378672
Relationships
  Objects [1, 2, 3, 4, 6, 7] front of: 0
  Objects [2, 3, 4] front of: 1
  Objects [3] front of: 2
  Objects [] front of: 3
```

Question: What is the shape of the object that is both on the right side of the brown cylinder and in front of the blue cylinder?

Figure 8.1: Example image, question, and partial scene graph from CLEVR (Johnson *et al.* [25]).

begins with a ”text” field, which defines the synthesized command’s basic forms, including optional parameters and words. A later step transforms the text to create a diverse set of commands using the same underlying program.

The second section of the template identifies the relational and physical properties of the objects and the path to be generated, which can take the values defined by Table 6.2. We add an affordance field that describes how a path regards a set of objects by modulating the potential fields used to generate the cost map. The third section of the template defines how object sets relate to each other. In this example, we exclude cases where the sets are identical to avoid degenerate or contradictory cases. The second constraint ensures that object set 2 is never empty. The attributes in the template are randomly assigned until the constraints are satisfied. Some properties may be null. For example, the shape, color, and material properties are optional in this example. A small percentage of the object sets do not exist in the scene.

The final section of the template defines the sequence of operations the script uses to generate a path. Early efforts to solve the CLEVR dataset, such as IEP ([17]), predict this program and then construct a modular network to generate an answer to the question and image. We retain the program as a means to create data but discard all intermediate results except the command and path.

The process generates random endpoints that do not coincide with any objects in the scene and constructs a potential field around the objects, as defined by its affordance property. We use A* and Field D* to generate a path with the cost map and a baseline shortest path that avoids obstacles using the Open Motion Planning Library (OMPL [131]).

```
{"text": ["Go <A>ing [the] (<Z1> <C1> <M1>) <S1> (along with) [the] (<Z2> <C2> <M2>) <S2>",
          "<A> [the] (<Z1> <C1> <M1>) <S1> (along with) [the] (<Z2> <C2> <M2>) <S2>"],
 "params": [{"type": "Size", "name": "<Z1>"},
            {"type": "Color", "name": "<C1>"},
            {"type": "Material", "name": "<M1>"},
            {"type": "Shape", "name": "<S1>"},
            {"type": "Affordance", "name": "<A>"},
            {"type": "Size", "name": "<Z2>"},
            {"type": "Color", "name": "<C2>"},
            {"type": "Material", "name": "<M2>"},
            {"type": "Shape", "name": "<S2>"}],
 "constraints": [{"params": {"set1": ["<Z1>","<C1>","<M1>","<S1>"],
                             "set2": ["<Z2>","<C2>","<M2>","<S2>"]}, "type": "NEQ_PARAM"},
                 {"params": {"set2": ["<Z2>","<C2>","<M2>","<S2>"]}, "type": "NOT_NULL_PARAM"}],
 "nodes":[{"inputs": [], "function": "scene"},
         {"inputs": ["scene"], "function": "random_endpoints"},
         {"inputs": ["scene"], "function": "filter", "name": "set1",
          "value_inputs": ["<Z1>", "<C1>", "<M1>", "<S1>"], "rank": 2},
         {"inputs": ["scene"], "function": "filter", "name": "set2",
          "value_inputs": ["<Z2>", "<C2>", "<M2>", "<S2>"], "rank": 2},
         {"inputs": ["set1", "set2"], "function": "union"},
         {"inputs": ["union"], "function": "affordance",
          "value_inputs": ["<A>"]},
         {"inputs": [], "function": "thunk"},
         {"inputs": ["affordance", "random_endpoints"], "function": "plan"}
        ],
 "tag": {"family": 0, "major": 0, "minor": 1}}
```

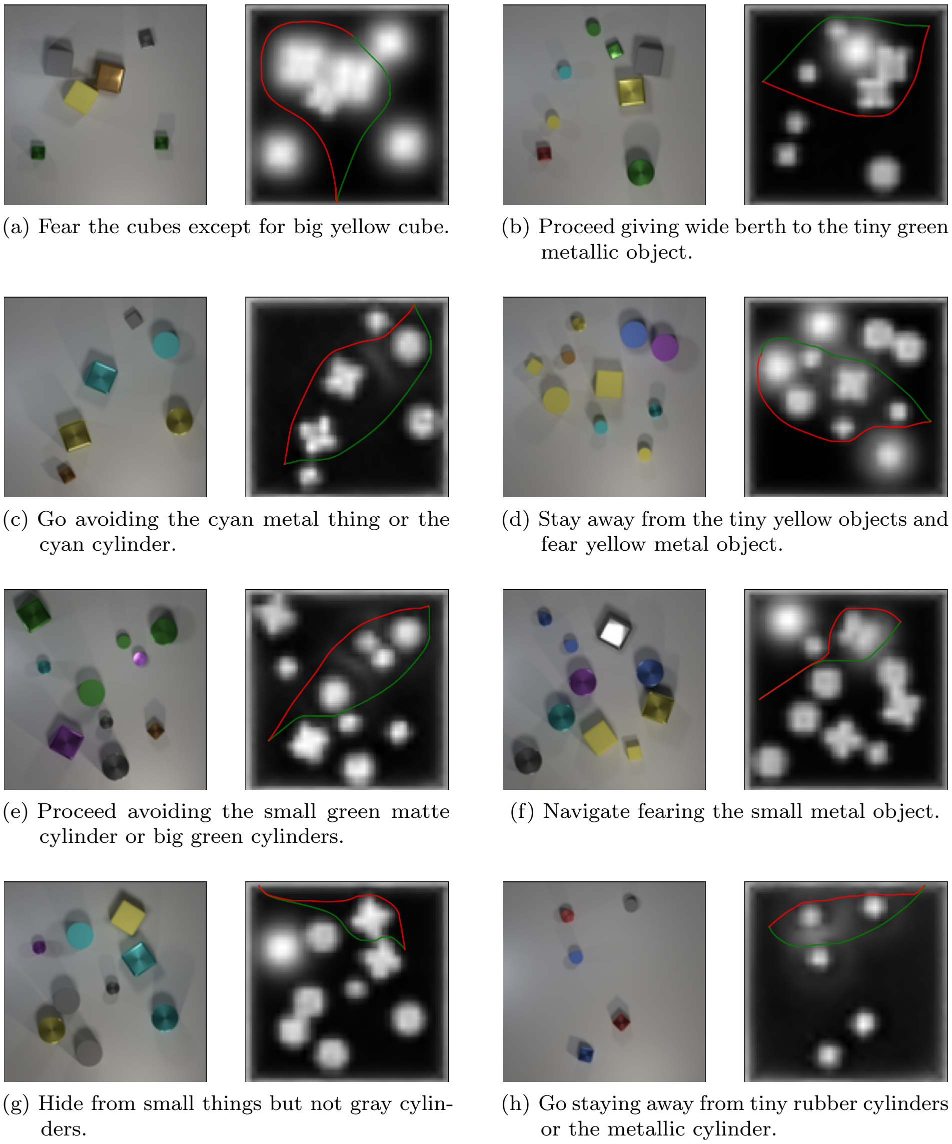
Figure 8.2: Sample template and program for command family 1.

8.2 Supplementary Results for Meta-Parameter Selection

This section contains the worst 12 examples from the validation set from the run shown in Section 6.1.1 to give some insight into the errors the algorithm makes.

Table 8.1: Statistics for Figure 8.3

Figure	Family	$^{90}F_{GT}$	$^{90}F_{SP}$	$^{90}H_{GT}$	$^{90}H_{SP}$	Loss
Figure 8.3a	3	65.14	14.27	33.67	4.87	1.8e-02
Figure 8.3b	0	61.74	22.75	34.00	13.16	1.6e-02
Figure 8.3c	4	53.97	45.94	33.43	33.96	1.8e-02
Figure 8.3d	2	53.33	15.04	33.88	6.00	5.1e-02
Figure 8.3e	4	42.33	38.18	26.16	23.32	1.7e-02
Figure 8.3f	0	32.75	5.11	4.00	2.24	7.0e-02
Figure 8.3g	3	28.63	7.09	6.95	1.52	1.7e-02
Figure 8.3h	4	28.05	29.52	18.56	20.81	1.7e-02
Figure 8.3i	4	27.90	4.85	10.92	3.93	2.4e-02
Figure 8.3j	4	27.69	26.15	15.43	13.55	2.4e-02
Figure 8.3k	2	24.76	5.66	6.90	2.63	1.7e-02
Figure 8.3l	2	21.82	22.31	5.41	4.82	2.1e-02



(a) Fear the cubes except for big yellow cube.

(b) Proceed giving wide berth to the tiny green metallic object.

(c) Go avoiding the cyan metal thing or the cyan cylinder.

(d) Stay away from the tiny yellow objects and fear yellow metal object.

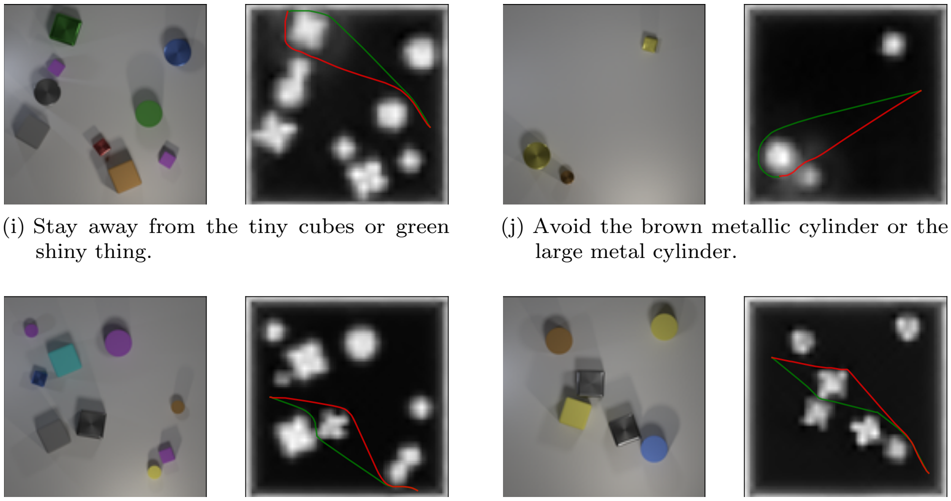
(e) Proceed avoiding the small green matte cylinder or big green cylinders.

(f) Navigate fearing the small metal object.

(g) Hide from small things but not gray cylinders.

(h) Go staying away from tiny rubber cylinders or the metallic cylinder.

Figure 8.3: Worst examples from sample of validation dataset, 4160 samples. Source image (left), generated cost map (right), ground truth path (green), generated path (red).



(i) Stay away from the tiny cubes or green shiny thing.

(j) Avoid the brown metallic cylinder or the large metal cylinder.

(k) Proceed trusting cyan rubber ball and trusting the metallic cubes.

(l) Trust the matte cylinders and trust blocks.

Figure 8.3: Worst examples from sample of validation dataset, 4160 samples. Source image (left), generated cost map (right), ground truth path (green), generated path (red).

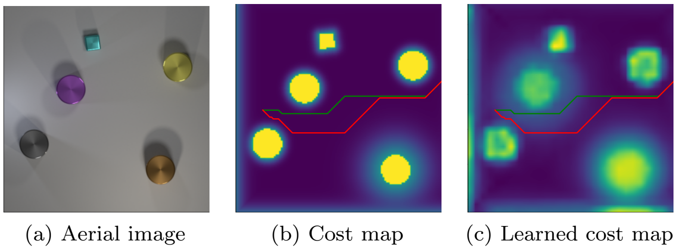
8.3 Supplementary Figures for Error Analysis

The following plots are worst examples from the analysis in Section 6.2. In all cases, the green path is the ground truth path and the red path is the path generated from the inferred cost map. These examples all use the A* path planner.

Table 8.2: Worst examples from each region, testing set. Region 1 paths follow the ground truth path and match cost. Region 2 paths follow the ground truth path but stray into high-cost regions. Region 3 paths choose alternative routes that have similar cost to the ground truth path. Region 4 paths deviate from the ground truth path and have substantially higher cost.

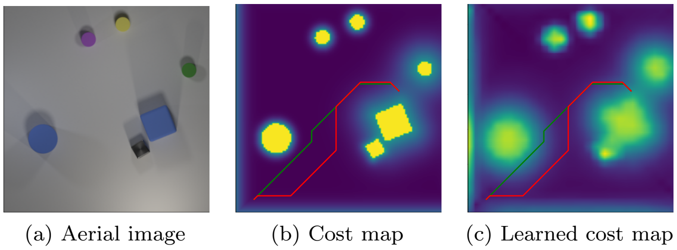
Region	Figure	${}^{90}F_{SP}$	Percent Cost Error
1	Figure 8.4	14.87	8.48
1	Figure 8.5	14.87	9.95
1	Figure 8.6	14.87	0.00
1	Figure 8.7	14.87	0.00
1	Figure 8.8	14.87	0.33
2	Figure 8.9	8.06	1027.53
2	Figure 8.10	4.12	874.94
2	Figure 8.11	5.39	811.46
2	Figure 8.12	2.00	796.84
2	Figure 8.13	2.00	781.00
3	Figure 8.14	97.00	1.21
3	Figure 8.15	95.34	4.78
3	Figure 8.16	89.10	0.42
3	Figure 8.17	88.14	6.42
3	Figure 8.18	88.00	12.95
4	Figure 8.19	100.41	29.19
4	Figure 8.20	100.00	28.34
4	Figure 8.21	97.19	21.88
4	Figure 8.22	94.00	22.24
4	Figure 8.23	90.62	19.12

8.3.1 Region 1



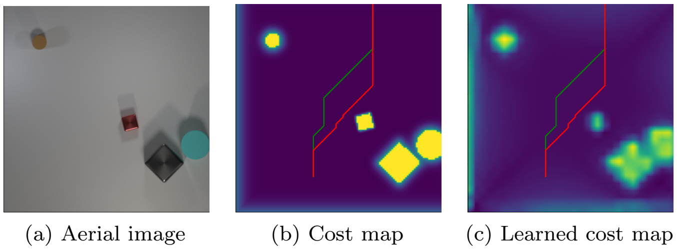
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.4: Command: “Give wide berth to the large brown cylinder as well as big blue cylinder.”



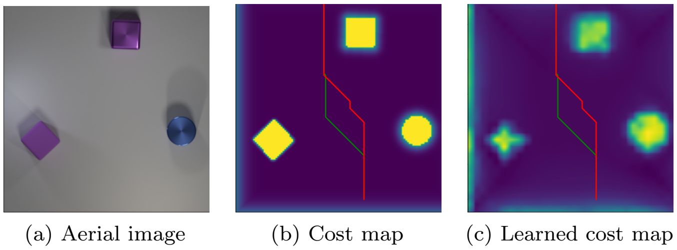
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.5: Command: “Move keeping away from small green cylinder and the large blue rubber block.”



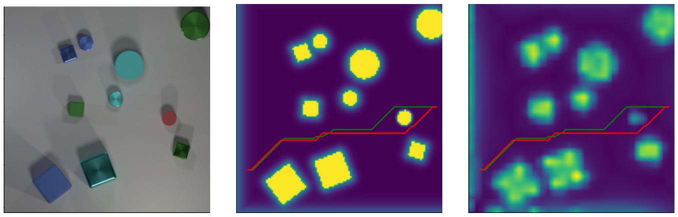
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.6: Command: “Trust small objects except for the tiny brown thing.”



(a) Aerial image (b) Cost map (c) Learned cost map

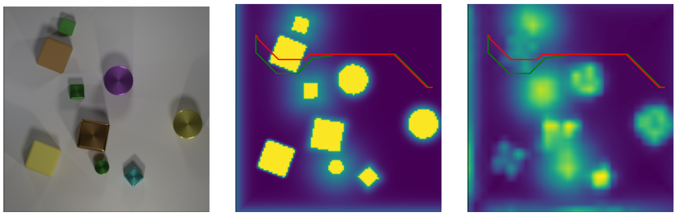
Figure 8.7: Command: “Proceed trusting the large rubber thing and trusting purple matte cube.”



(a) Aerial image (b) Cost map (c) Learned cost map

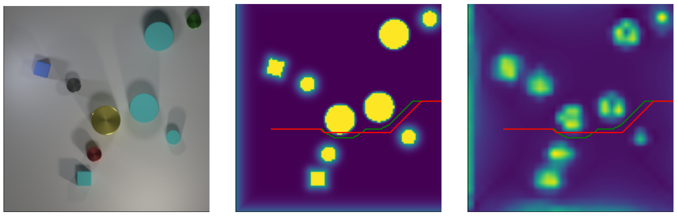
Figure 8.8: Command: “Navigate trusting the red object.”

8.3.2 Region 2



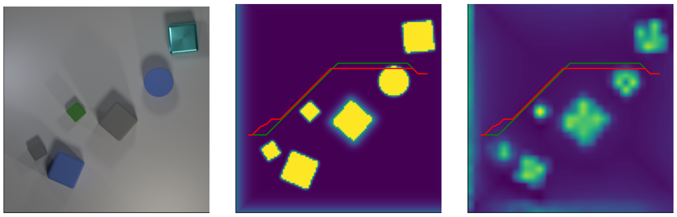
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.9: Command: “Trust big rubber things and hide from green things.”



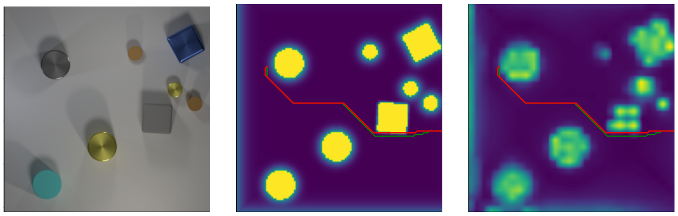
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.10: Command: “Move trusting the large cyan cylinders in addition to the yellow cylinder.”



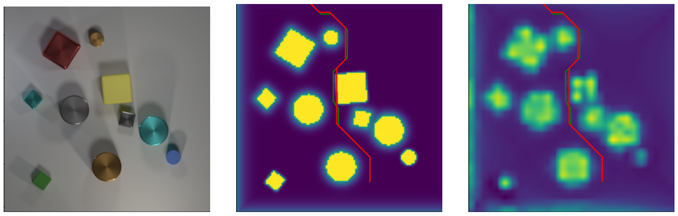
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.11: Command: “Navigate trusting objects except for big gray matte thing.”



(a) Aerial image (b) Cost map (c) Learned cost map

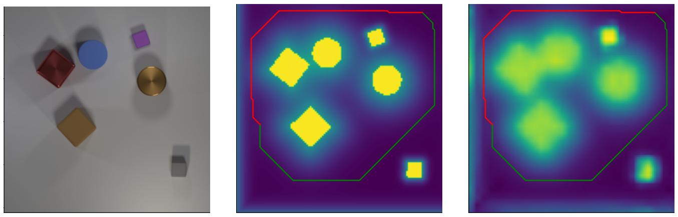
Figure 8.12: Command: “Trust the matte objects except matte cylinders.”



(a) Aerial image (b) Cost map (c) Learned cost map

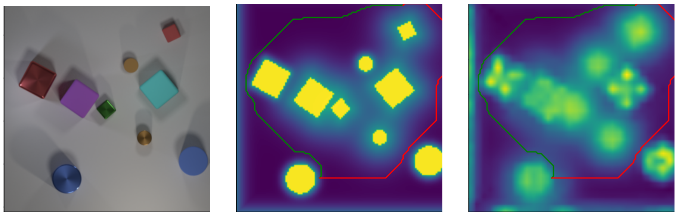
Figure 8.13: Command: “Proceed trusting matte things.”

8.3.3 Region 3



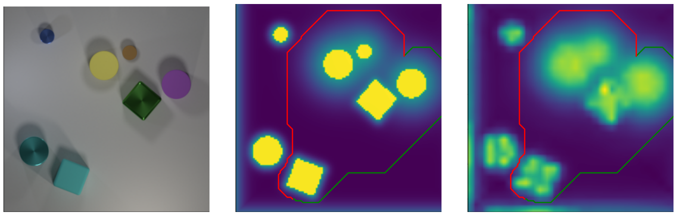
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.14: Command: “Stay away from big things and blue thing.”



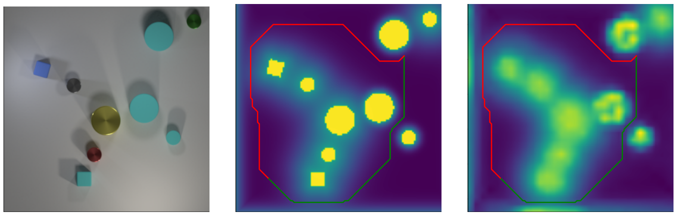
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.15: Command: “Stay away from rubber cubes as well as tiny shiny things.”



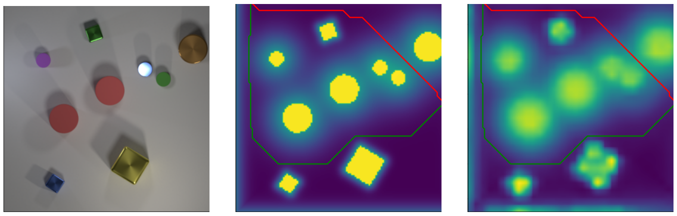
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.16: Command: “Keep away from matte cylinders.”



(a) Aerial image (b) Cost map (c) Learned cost map

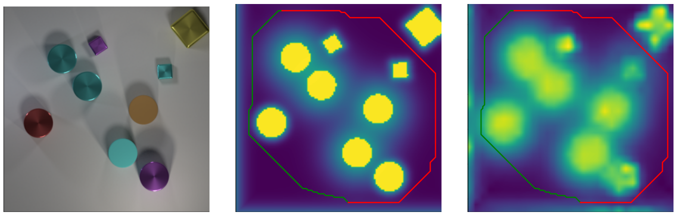
Figure 8.17: Command: “Hide from shiny things including the blocks.”



(a) Aerial image (b) Cost map (c) Learned cost map

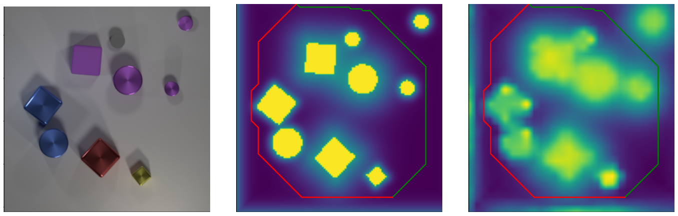
Figure 8.18: Command: “Navigate hiding from the red matte cylinders along with the cylinders.”

8.3.4 Region 4



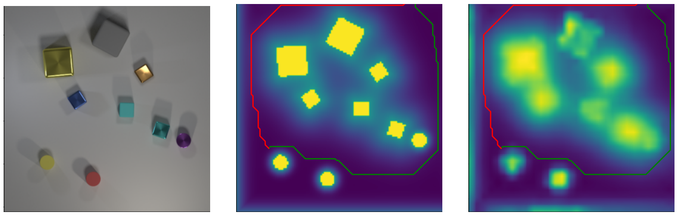
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.19: Command: “Stay away from big cyan things as well as brown thing.”



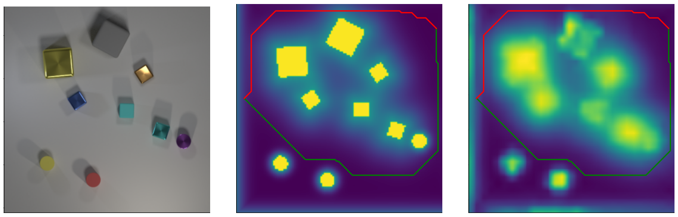
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.20: Command: “Proceed avoiding big purple things and hiding from the red metal cube.”



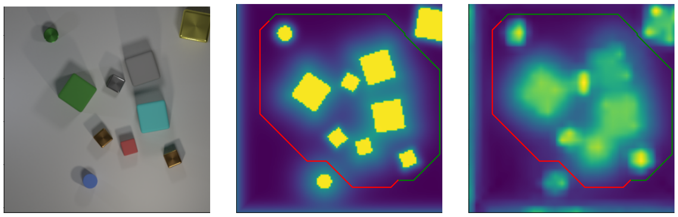
(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.21: Command: “Move staying away from the big metal object and fearing blocks.”



(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.22: Command: “Move staying away from the big metal object and fearing blocks.”

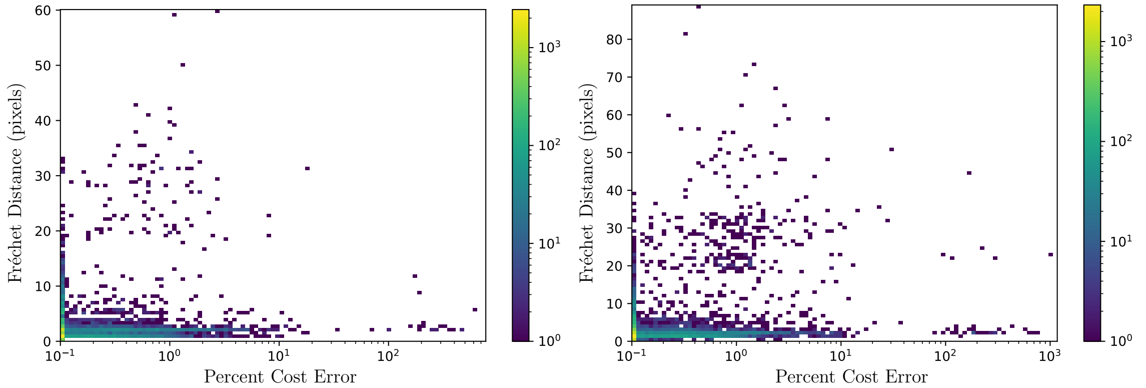


(a) Aerial image (b) Cost map (c) Learned cost map

Figure 8.23: Command: “Fear rubber blocks and the tiny rubber things.”

8.4 Supplementary Histograms for Error Analysis

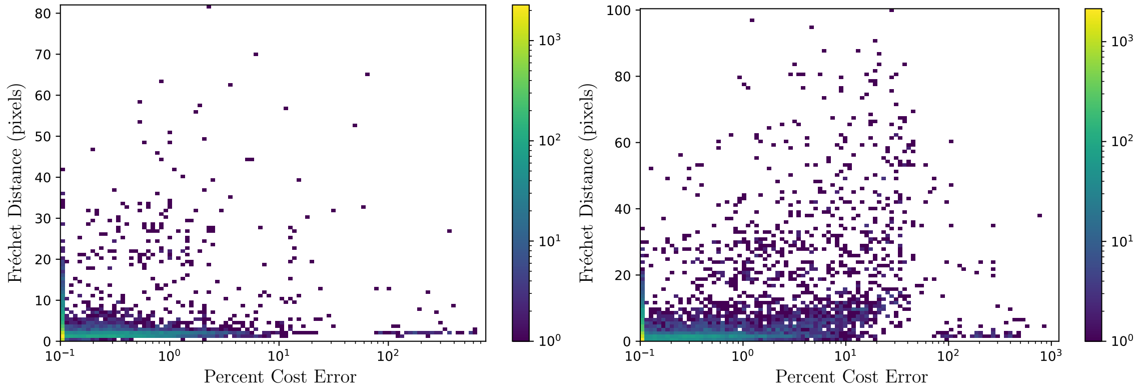
This section contains by-family histograms for the various command families as part of the investigation in Section 6.2. Examples of outlier paths for each family and region can be found in Section 8.3.



(a) Train, family 0, linear-log.

(b) Test, family 0, linear-log.

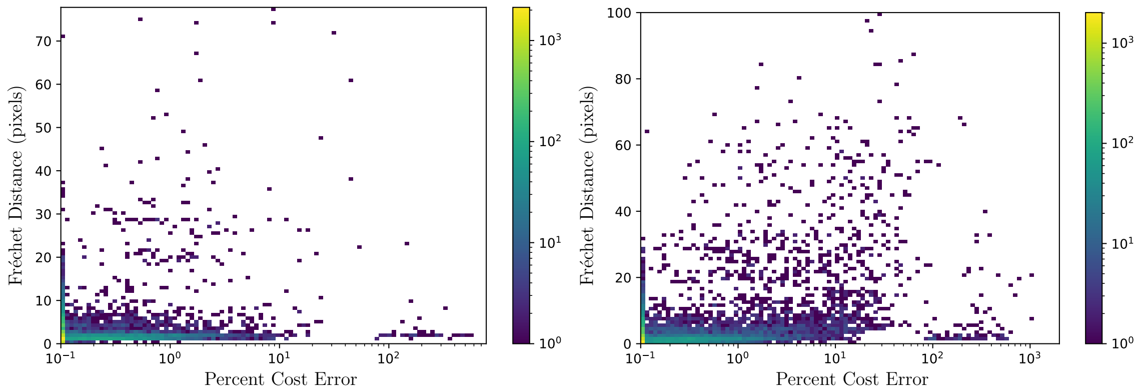
Figure 8.24: Error histograms for train and test dataset, command family 0.



(a) Train, family 1, linear-log.

(b) Test, family 1, linear-log.

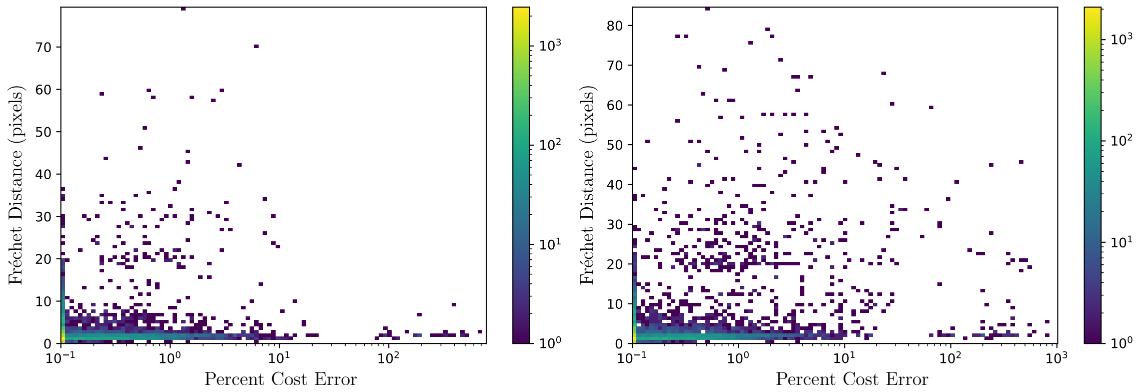
Figure 8.25: Error histograms for train and test dataset, command family 1.



(a) Train, family 2, linear-log.

(b) Test, family 2, linear-log.

Figure 8.26: Error histograms for train and test dataset, command family 2.



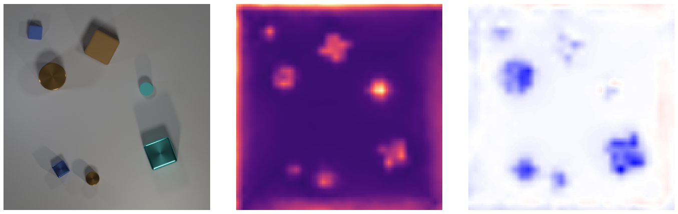
(a) Train, family 3, linear-log.

(b) Test, family 3, linear-log.

Figure 8.27: Error histograms for train and test dataset, command family 3.

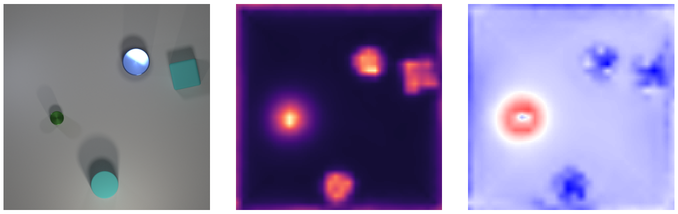
8.5 Supplementary Cost Map Modulation Examples

This section contains additional examples of cost map modulation from Section 6.3.1.



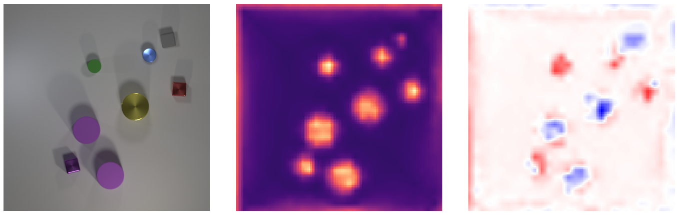
(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.28: Command: "Move trusting the large cylinder and trusting the cubes." Command family: 2.



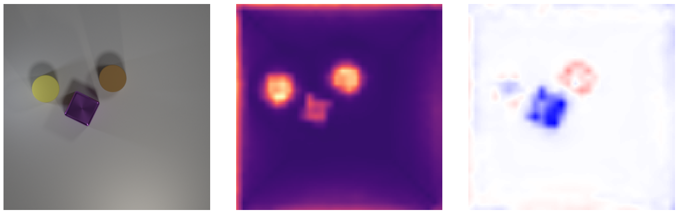
(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.29: Command: “Keep away from metal cylinders but not the large metal cylinder.” Command family: 3.



(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.30: Command: “Navigate trusting the small matte block.” Command family: 0.



(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.31: Command: “Trust the large shiny object and trust the big purple metal object.” Command family: 2.

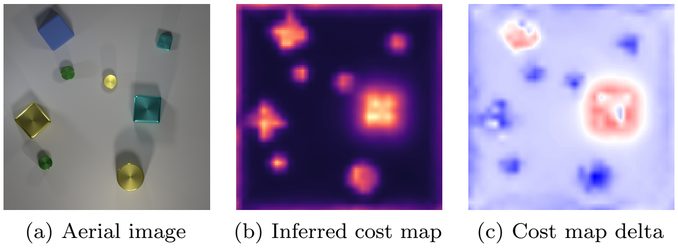

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.32: Command: "Go staying away from shiny cubes but not big yellow shiny things." Command family: 3.

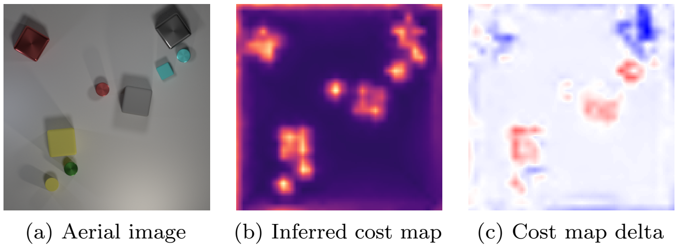

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.33: Command: "Trust big gray things except big gray matte object." Command family: 3.

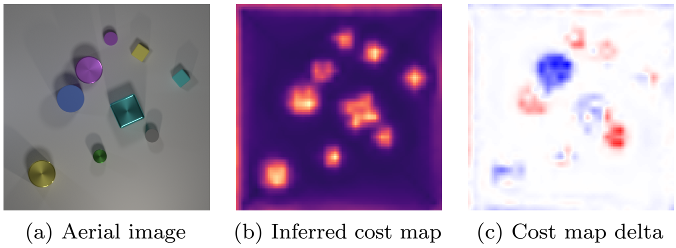

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.34: Command: "Go trusting big purple metal cylinder." Command family: 0.

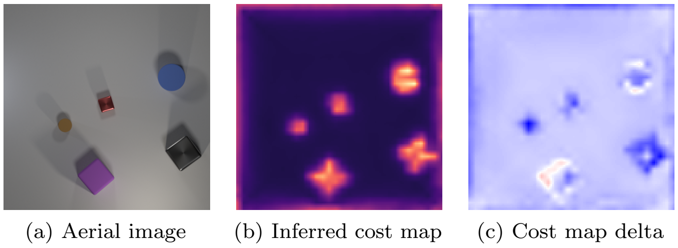

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.35: Command: "Trust the large brown rubber block and give wide berth to large red metal cylinder." Command family: 2.

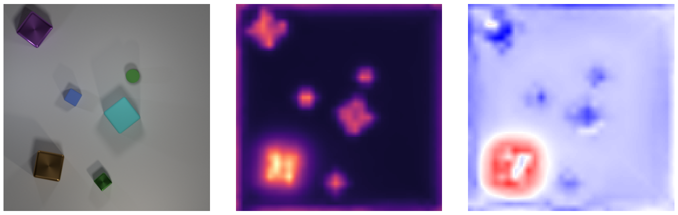

(a) Aerial image (b) Inferred cost map (c) Cost map delta

Figure 8.36: Command: “Avoid the small green rubber thing and stay away from the brown thing.” Command family: 2.

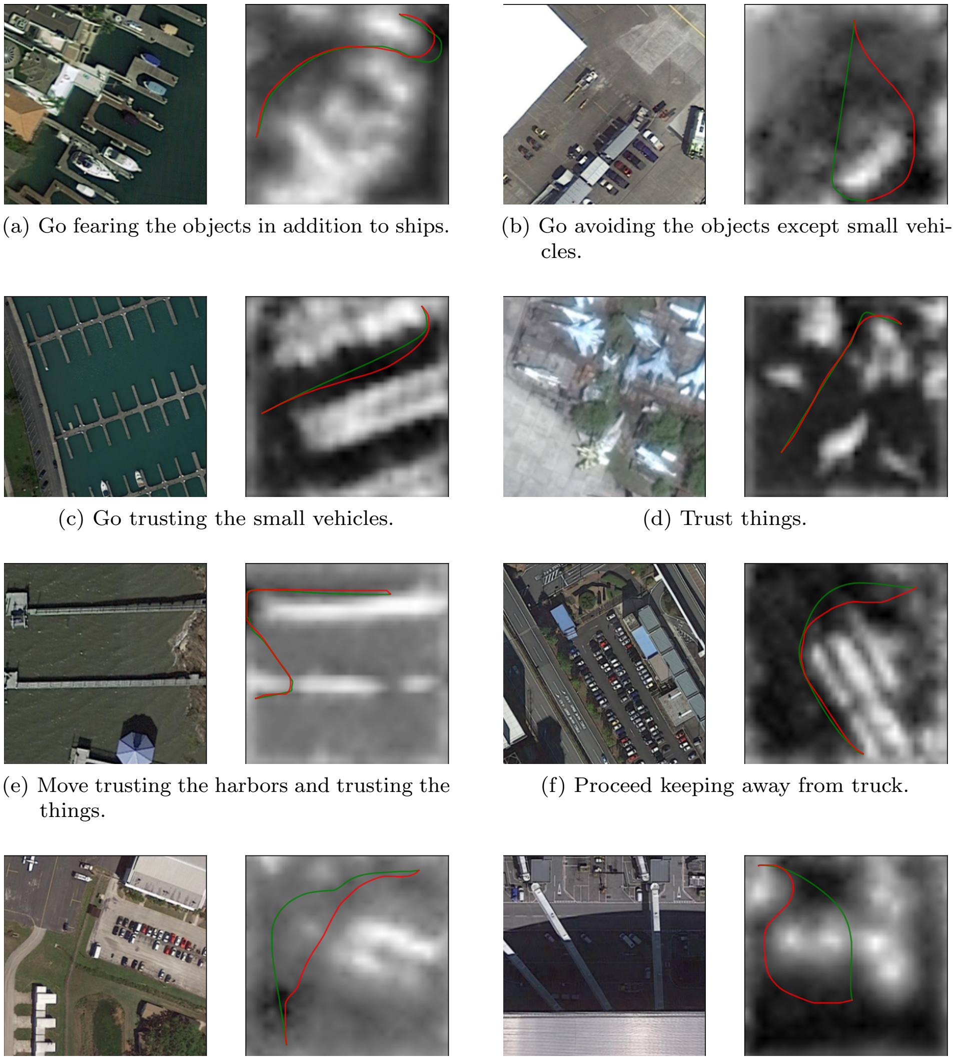
8.6 Supplementary Results for DOTA/iSAID Experiments

8.6.1 Training Dataset

This section contains the worst examples from a sample ($N = 4096$) of the training dataset.

Table 8.3: Statistics for Figure 8.37. Worst examples from training set.

Figure	Family	${}^{90}F_{GT}$	${}^{90}F_{SP}$	${}^{90}H_{GT}$	${}^{90}H_{SP}$	Loss
Figure 8.37a	1	14.20	42.71	2.68	13.30	7.16e-03
Figure 8.37b	3	99.16	23.74	49.01	11.37	7.65e-03
Figure 8.37c	0	14.24	10.51	5.59	3.72	7.34e-03
Figure 8.37d	0	7.20	3.96	1.41	1.00	7.72e-03
Figure 8.37e	2	5.00	137.57	1.36	118.02	8.30e-03
Figure 8.37f	0	24.42	55.61	3.77	38.29	7.42e-03
Figure 8.37g	3	63.29	129.36	17.44	74.65	7.55e-03
Figure 8.37h	3	110.00	53.43	52.84	37.52	9.05e-03
Figure 8.37i	2	10.37	12.17	6.27	2.28	1.07e-02
Figure 8.37j	0	13.76	71.24	2.14	59.68	8.38e-03
Figure 8.37k	2	120.49	30.46	62.65	19.65	1.68e-02
Figure 8.37l	2	58.71	12.99	23.77	2.48	1.27e-02
Figure 8.37m	2	32.11	46.79	2.92	21.64	9.96e-03
Figure 8.37n	2	7.75	17.25	0.79	5.81	8.87e-03
Figure 8.37o	2	55.14	3.71	32.68	1.41	1.07e-02
Figure 8.37p	2	44.94	48.97	4.07	11.00	1.18e-02



(a) Go fearing the objects in addition to ships.

(b) Go avoiding the objects except small vehicles.

(c) Go trusting the small vehicles.

(d) Trust things.

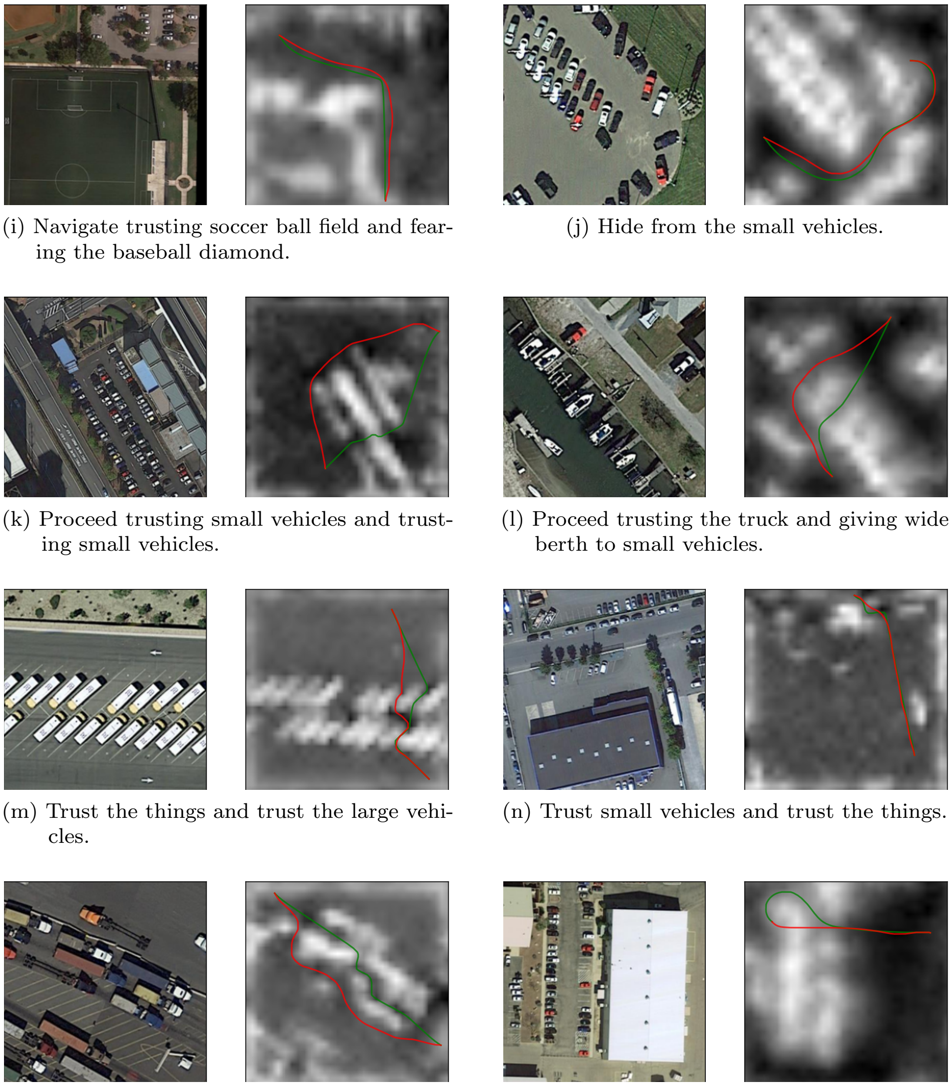
(e) Move trusting the harbors and trusting the things.

(f) Proceed keeping away from truck.

(g) Give wide berth to objects except for the large vehicles.

(h) Go staying away from things excluding the large vehicle.

Figure 8.37: Worst examples from sample of training dataset. Source image (left), generated cost map (right), ground truth path (green), generated path (red).



(i) Navigate trusting soccer ball field and fearing the baseball diamond.

(j) Hide from the small vehicles.

(k) Proceed trusting small vehicles and trusting small vehicles.

(l) Proceed trusting the truck and giving wide berth to small vehicles.

(m) Trust the things and trust the large vehicles.

(n) Trust small vehicles and trust the things.

(o) Trust the large vehicles and trust the large vehicles.

(p) Go fearing small vehicles and giving wide berth to small vehicles.

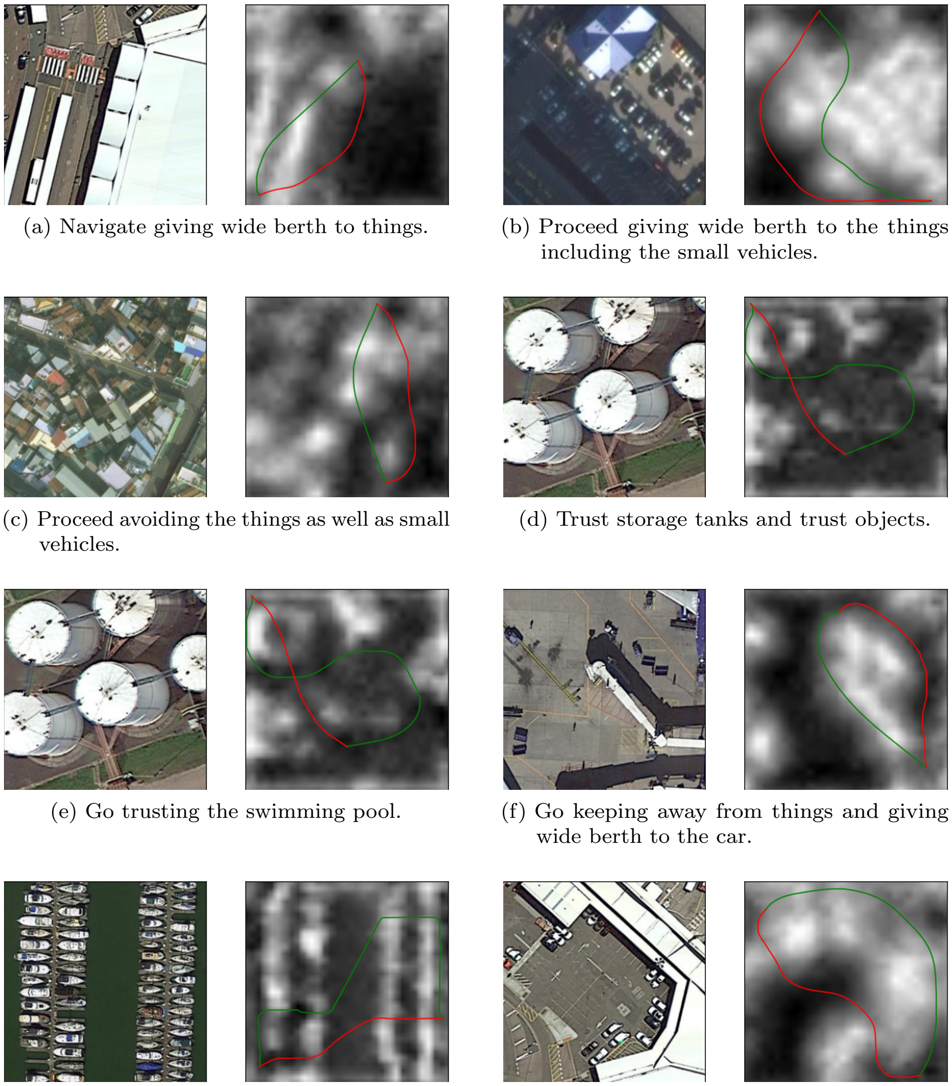
Figure 8.37: Worst examples from sample of training dataset. Source image (left), generated cost map (right), ground truth path (green), generated path (red).

8.6.2 Validation Dataset

This section contains the worst examples from a sample ($N = 4096$) of the validation dataset.

Table 8.4: Supplementary information for Figure 8.38. Worst examples from validation set.

Figure	Family	${}^{90}F_{GT}$	${}^{90}F_{SP}$	${}^{90}H_{GT}$	${}^{90}H_{SP}$	Loss
Figure 8.38a	0	72.21	19.24	53.78	11.31	1.24e-01
Figure 8.38b	1	82.72	66.76	56.39	30.13	1.27e-01
Figure 8.38c	1	69.00	34.63	53.17	21.93	1.26e-01
Figure 8.38d	2	123.63	68.26	42.82	3.13	1.51e-01
Figure 8.38e	0	130.00	13.69	46.01	8.12	1.40e-01
Figure 8.38f	2	108.52	45.49	58.59	32.14	1.29e-01
Figure 8.38g	0	128.17	231.99	61.01	134.35	1.69e-01
Figure 8.38h	1	132.74	116.04	93.25	85.14	1.51e-01
Figure 8.38i	2	29.10	7.74	25.42	3.21	1.54e-01
Figure 8.38j	0	32.29	5.05	15.67	2.57	2.67e-01
Figure 8.38k	0	143.06	123.47	69.05	55.36	1.41e-01
Figure 8.38l	2	110.99	79.92	52.97	11.75	1.51e-01
Figure 8.38m	1	131.13	116.40	74.27	60.01	1.48e-01
Figure 8.38n	2	202.56	214.56	119.93	156.32	1.70e-01
Figure 8.38o	2	180.69	207.71	90.07	160.86	2.09e-01
Figure 8.38p	2	44.43	14.18	27.15	8.23	1.61e-01



(a) Navigate giving wide berth to things.

(b) Proceed giving wide berth to the things including the small vehicles.

(c) Proceed avoiding the things as well as small vehicles.

(d) Trust storage tanks and trust objects.

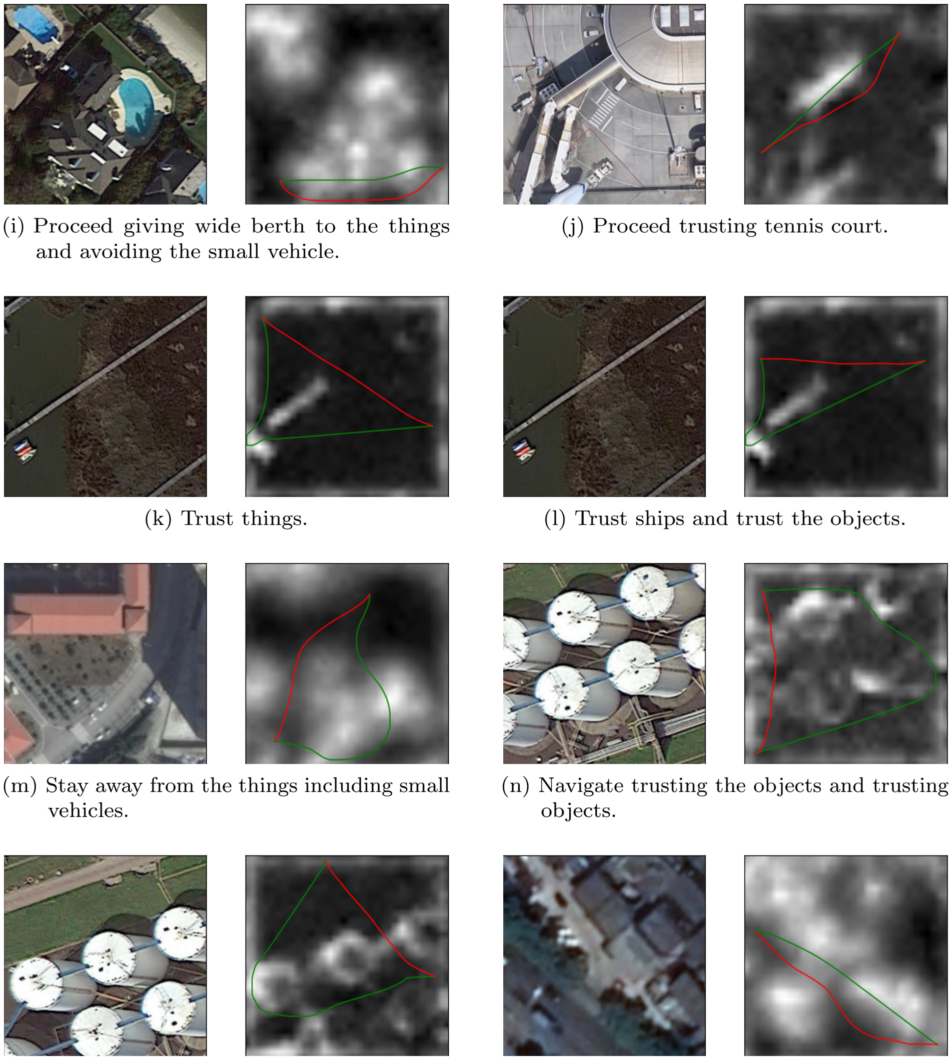
(e) Go trusting the swimming pool.

(f) Go keeping away from things and giving wide berth to the car.

(g) Navigate trusting objects.

(h) Navigate giving wide berth to objects and small vehicles.

Figure 8.38: Worst examples from sample of validation dataset. Source image (left), generated cost map (right), ground truth path (green), generated path (red).



(i) Proceed giving wide berth to the things and avoiding the small vehicle.

(j) Proceed trusting tennis court.

(k) Trust things.

(l) Trust ships and trust the objects.

(m) Stay away from the things including small vehicles.

(n) Navigate trusting the objects and trusting objects.

(o) Navigate trusting the objects and trusting the storage tanks.

(p) Keep away from the things and give wide berth to small vehicles.

Figure 8.38: Worst examples from sample of validation dataset. Source image (left), generated cost map (right), ground truth path (green), generated path (red).

8.7 Publications

This section contains a list of related publications which include the author.

[1] A. J. Suppé and M. Hebert, “Using deep learning to bridge the gap between perception and intelligence,” in *Unmanned Systems Technology XIX*, International Society for Optics and Photonics, vol. 10195, 2017, p. 1 019 503.

[2] M. Childers, C. Lennon, B. Bodt, J. Pusey, S. Hill, R. Camden, J. Oh, R. Dean, T. Keegan, C. Diberardino, S. Karumanchi, B. Douillard, N. Gupta, C. Ordonez, J. Shill, E. Collins, J. Clark, D. Barber, J. Duperret, L. Navarro-Serment, and A. Suppé, “US Army Research Laboratory (ARL) Robotics Collaborative Technology Alliance 2014 Capstone Experiment,” US Army Research Laboratory Aberdeen Proving Ground United States, Tech. Rep., 2016.

[3] J. H. Oh, M. Zhu, S. Park, T. M. Howard, M. R. Walter, D. Barber, O. Romero, A. Suppé, L. E. Navarro-Serment, F. Duvallet, A. Boularias, J. Vinokurov, T. Keegan, R. Dean, C. Lennon, B. Bodt, M. Childers, J. Shi, K. Daniilidis, N. Roy, C. Lebiere, M. Hebert, and A. Stentz, “Integrated intelligence for human-robot teams,” in *Proceedings of International Symposium on Experimental Robotics (ISER ’16)*, Oct. 2016, pp. 309–322.

[4] M. Wigness, J. G. Rogers, L. E. Navarro-Serment, A. Suppé, and B. A. Draper, “Reducing adaptation latency for multi-concept visual perception in outdoor environments,” in *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2016, pp. 2784–2791.

[5] C. Lennon, B. Bodt, M. Childers, J. Oh, A. Suppé, L. Navarro-Serment, R. Dean, T. Keegan, C. Diberardino, and M. Zhu, “An integrated assessment of progress in robotic perception and semantic navigation,” US Army Research Laboratory Aberdeen Proving Ground United States, Tech. Rep., 2015.

[6] J. H. Oh, A. Suppé, F. Duvallet, A. Boularias, L. Navarro-Serment, M. Hebert, A. Stentz, J. Vinokurov, O. Romero, C. Lebiere, *et al.*, “Toward mobile robots reasoning like humans,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[7] J. H. Oh, L. E. Navarro-Serment, A. Suppé, A. Stentz, and M. Hebert, “Inferring door locations from a teammate’s trajectory in stealth human-robot team operations,” in *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2015, pp. 5315–5320.

[8] C. Lennon, B. Bodt, M. Childers, R. Camden, A. Suppé, L. Navarro-Serment, and N. Florea, “Performance evaluation of a semantic perception classifier,” US Army Research Laboratory Aberdeen Proving Ground United States, Tech. Rep., 2013.

[9] J. H. Oh, A. Suppé, A. Stentz, and M. Hebert, “Enhancing robot perception using human teammates,” in *Proceedings of International Conference on Autonomous Agents and MultiAgent Systems (AAMAS ’13)*, May 2013, pp. 1147–1148.

[10] A. Suppé, L. E. Navarro-Serment, D. Munoz, J. A. (Bagnell, and M. Hebert, “An architecture for online semantic labeling on UGVs,” in *Proceedings of SPIE Unmanned Systems Technology XV*, R. E. Karlsen, D. W. Gage, C. M. Shoemaker, and G. R. Gerhart, Eds., vol. 8741, Apr. 2013.

Bibliography

[1] J. H. Oh, A. Suppé, F. Duvallet, A. Boularias, L. E. Navarro-Serment, M. Hebert, A. Stentz, J. Vinokurov, O. J. Romero, C. Lebiere, and R. Dean, "Toward mobile robots reasoning like humans," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 2015, pp. 1371–1379.

[2] B. Zhou, Y. Tian, S. Sukhbaatar, A. Szlam, and R. Fergus, "Simple baseline for visual question answering," *CoRR*, vol. abs/1512.02167, 2015.

[3] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 21–29.

[4] N. J. Nilsson, "Shakey the robot," SRI International, Tech. Rep. 323, Apr. 1984.

[5] H. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Mag.*, vol. 9, no. 2, pp. 61–74, Jul. 1988.

[6] S. Thrun, "Particle filters in robotics," in *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[8] C. B. Congdon, M. J. Huber, D. Kortenkamp, K. Konolige, K. L. Myers, A. Saffiotti, and E. H. Ruspini, "Carmel versus flakey: A comparison of two winners," *AI Magazine*, vol. 14, no. 1, pp. 49–57, 1993.

[9] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. ". Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[10] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* New York, NY, USA: Henry Holt and Co., Inc., 1982.

[11] J. Oh, T. M. Howard, M. R. Walter, D. Barber, M. Zhu, Z. Park, A. Suppé, L. Navarro-Serment, F. Duvallet, A. Boularias, O. Romero, J. Vinokurov, T. Keegan, R. Dean, craig Lennon, B. Bodt, M. Childers, J. Shi, K. Daniilidis, N. Roy, C. Lebiere, M. Hebert, and A. Stentz, "Integrated intelligence for human-robot teams," in *Proceedings of the International Symposium on Experimental Robotics (ISER)*, Tokyo, Japan, Oct. 2016.

[12] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis, "Single image 3d object detection and pose estimation for grasping," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3936–3943.

[13] D. Munoz, "Inference machines: Parsing scenes via iterated predictions," Ph.D. dissertation, The Robotics Institute, Carnegie Mellon University, Jun. 2013.

[14] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Mar. 2010, pp. 259–266.

[15] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[16] L. Chen, H. Zhang, J. Xiao, L. Nie, J. Shao, W. Liu, and T.-S. Chua, "SCA-CNN: Spatial and channel-wise attention in convolutional networks for image captioning," in *CVPR*, 2017.

[17] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick, "Inferring and executing programs for visual reasoning," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 3008–3017.

[18] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 2154–2162.

[19] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. Van Den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3674–3683.

[20] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06, Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 729–736.

[21] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, 2008, pp. 1433–1438.

[22] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[23] J. Lu, J. Yang, D. Batra, and D. Parikh, "Hierarchical question-image co-attention for visual question answering," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 289–297.

[24] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville, "FiLM: visual reasoning with a general conditioning layer," in *AAAI*, 2018.

[25] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick, "CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 1988–1997.

[26] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[27] J. Ding, N. Xue, Y. Long, G.-S. Xia, and Q. Lu, “Learning roi transformer for detecting oriented objects in aerial images,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[28] J. Ding, N. Xue, G.-S. Xia, X. Bai, W. Yang, M. Y. Yang, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, *Object detection in aerial images: A large-scale benchmark and challenges*, 2021.

[29] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, “DOTA: A large-scale dataset for object detection in aerial images,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[30] S. W. Zamir, A. Arora, A. Gupta, S. Khan, G. Sun, F. Shahbaz Khan, F. Zhu, L. Shao, G.-S. Xia, and X. Bai, “ISAID: A large-scale dataset for instance segmentation in aerial images,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 28–37.

[31] S. Harnad, “The symbol grounding problem,” *Physica D*, vol. 42, pp. 335–346, 1990.

[32] M. MacMahon, B. Stankiewicz, and B. Kuipers, “Walk the talk: Connecting language, knowledge, and action in route instructions,” in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, ser. AAAI’06, AAAI Press, 2006, pp. 1475–1482.

[33] A. Boularias, F. Duvallet, J. Oh, and A. Stentz, “Grounding spatial relations for outdoor robot navigation,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1976–1982.

[34] S. Guadarrama, L. Riano, D. Golland, D. Göhring, Y. Jia, D. Klein, P. Abbeel, and T. Darrell, “Grounding spatial relations for human-robot interaction,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 1640–1647.

[35] S. Hemachandra, F. Duvallet, T. M. Howard, N. Roy, A. Stentz, and M. R. Walter, “Learning models for following natural language directions in unknown environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015.

[36] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, “Cognitive mapping and planning for visual navigation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[37] K. Chen, J. K. Chen, J. Chuang, M. Vázquez, and S. Savarese, “Topological planning with transformers for vision-and-language navigation,” *CoRR*, vol. abs/2012.05292, 2020.

[38] T. Wang, V. Dhiman, and N. Atanasov, “Inverse reinforcement learning for autonomous navigation via differentiable semantic mapping and planning,” *CoRR*, vol. abs/2101.00186, 2021.

[39] M. Wulfmeier, P. Ondrúška, and I. Posner, “Maximum entropy deep inverse reinforcement learning,” in *Neural Information Processing Systems Conference, Deep Reinforcement Learning Workshop*, vol. abs/1507.04888, Montreal, Canada, 2015.

[40] Y. Song, “Inverse reinforcement learning for autonomous ground navigation using aerial and satellite observation data,” M.S. thesis, Carnegie Mellon University, Pittsburgh, PA, May 2019.

[41] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *Proceedings of the 5th ACM/IEEE International Conference on Human-robot Interaction*, ser. HRI '10, Osaka, Japan: IEEE Press, 2010, pp. 259–266.

[42] D. Misra, J. Langford, and Y. Artzi, "Mapping instructions and visual observations to actions with reinforcement learning," pp. 1004–1015, Sep. 2017.

[43] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, p. 0 278 364 917 722 396, 2017.

[44] J. H. Oh, L. E. Navarro-Serment, A. Suppé, A. Stentz, and M. Hebert, "Inferring door locations from a teammate's trajectory in stealth human-robot team operations," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2015.

[45] A. Suppé, L. E. Navarro-Serment, D. Munoz, J. A. (Bagnell, and M. Hebert, "An architecture for online semantic labeling on ugvs," in *Proc. SPIE 8741, Unmanned Systems Technology XV*, R. E. Karlsen, D. W. Gage, C. M. Shoemaker, and G. R. Gerhart, Eds., Apr. 2013.

[46] J. H. Oh, A. Suppé, A. Stentz, and M. Hebert, "Enhancing robot perception using human teammates," in *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, May 2013.

[47] C. Lennon, B. Bodt, M. Childers, R. Dean, J. Oh, C. DiBerardino, and T. Keegan, "RCTA capstone assessment," in *Proc. SPIE*, vol. 9468, 2015.

[48] Y. Yang and D. Ramanan, "Articulated pose estimation with flexible mixtures-of-parts," in *CVPR 2011*, Jun. 2011, pp. 1385–1392.

[49] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[50] T. Ojala, M. Pietikainen, and D. Harwood, "Performance evaluation of texture measures with classification based on kullback discrimination of distributions," in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, Oct. 1994, 582–585 vol.1.

[51] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context," *Int. Journal of Computer Vision (IJCV)*, Jan. 2009.

[52] F. Duvallet, "Natural language direction following for robots in unstructured unknown environments," Ph.D. dissertation, School of Computer Science, Jan. 2015.

[53] A. Boularias, F. Duvallet, J. Oh, and A. Stentz, "Learning qualitative spatial relations for robotic navigation.," in *IJCAI*, 2016, pp. 4130–4134.

[54] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions," in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 1–8.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR09*, 2009.

[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[57] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04, Banff, Alberta, Canada: ACM, 2004, pp. 1–.

[58] N. Ratliff, "Learning to search: Structured prediction techniques for imitation learning," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2009.

[59] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," Ph.D. dissertation, School of Computer Science, 2010.

[60] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *Proc. of the International Conference on Intelligent Robots and Systems*, 2009.

[61] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *in Proc. 17th International Conf. on Machine Learning*, 2000, pp. 663–670.

[62] E. T. Jaynes, "Information theory and statistical mechanics," in *Physical Review*, vol. 106, 1957, pp. 620–630.

[63] N. Lee and K. M. Kitani, "Predicting wide receiver trajectories in American football," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2016, pp. 1–9.

[64] D.-A. Huang, A.-m. Farahmand, K. M. Kitani, and J. A. Bagnell, "Approximate maxent inverse optimal control and its application for mental simulation of human interactions," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15, Austin, Texas: AAAI Press, 2015, pp. 2673–2679.

[65] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 49–58.

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

[67] D. Mascharka, P. Tran, R. Soklaski, and A. Majumdar, "Transparency by design: Closing the gap between performance and interpretability in visual reasoning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[68] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *Int. J. Comput. Vision*, vol. 123, no. 1, pp. 32–73, May 2017.

[69] Yu Jiang, Vivek Natarajan, Xinlei Chen, M. Rohrbach, D. Batra, and D. Parikh, "Pythia v0.1: The winning entry to the vqa challenge 2018," *arXiv preprint arXiv:1807.09956*, 2018.

[70] Z. Yu, J. Yu, C. Xiang, J. Fan, and D. Tao, "Beyond bilinear: Generalized multimodal factorized high-order pooling for visual question answering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5947–5959, 2018.

[71] Z. Yu, J. Yu, J. Fan, and D. Tao, "Multi-modal factorized bilinear pooling with co-attention learning for visual question answering," *IEEE International Conference on Computer Vision (ICCV)*, pp. 1839–1848, 2017.

[72] J.-H. Kim, J. Jun, and B.-T. Zhang, "Bilinear attention networks," NIPS'18, pp. 1571–1581, 2018.

[73] D.-K. Nguyen and T. Okatani, "Improved fusion of visual and language representations by dense symmetric co-attention for visual question answering," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[74] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. B. Tenenbaum, "Neural-symbolic VQA: disentangling reasoning from vision and language understanding," *CoRR*, vol. abs/1810.02338, 2018.

[75] J. Suarez, J. Johnson, and F. Li, "DDRprog: A CLEVR differentiable dynamic reasoning programmer," *CoRR*, vol. abs/1803.11361, 2018.

[76] D. A. Hudson and C. D. Manning, "Compositional attention networks for machine reasoning," 2018.

[77] A. Agrawal, J. Lu, S. Antol, M. Mitchell, C. L. Zitnick, D. Batra, and D. Parikh, "VQA: Visual Question Answering," in *International Conference on Computer Vision (ICCV)*, 2015.

[78] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European Conference on Computer Vision (ECCV)*, Zürich, Jan. 1, 2014.

[79] Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei, "Visual7w: Grounded question answering in images," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 4995–5004.

[80] M. Malinowski and M. Fritz, "A multi-world approach to question answering about real-world scenes based on uncertain input," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 1682–1690.

[81] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, "Multimodal compact bilinear pooling for visual question answering and visual grounding," *EMNLP*, 2016.

[82] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778.

[83] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 1–9.

[84] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[85] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[86] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *ICLR*, 2013.

[87] A. Graves, “Generating sequences with recurrent neural networks,” *CoRR*, vol. abs/1308.0850, 2013.

[88] C. Xiong, S. Merity, and R. Socher, “Dynamic memory networks for visual and textual question answering,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML’16, New York, NY, USA: JMLR.org, 2016, pp. 2397–2406.

[89] M. P. Dubuisson and A. K. Jain, “A modified Hausdorff distance for object matching,” in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, Oct. 1994, 566–568 vol.1.

[90] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, “Comparing images using the Hausdorff distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850–863, Sep. 1993.

[91] M. D. Shapiro and M. B. Blaschko, “On Hausdorff distance metrics,” University of Massachusetts, Amherst, MA, Tech. Rep. UM-CS-2004-071, 2004.

[92] T. Eiter and H. Mannila, “Computing discrete Fréchet distance,” Tech. Rep., 1994.

[93] H. Alt and M. Godau, “Computing the Fréchet distance between two polygonal curves,” *Int. J. Comput. Geom. Appl.*, vol. 5, pp. 75–91, 1995.

[94] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir, “Computing the discrete Fréchet distance in subquadratic time,” *SIAM Journal on Computing*, vol. 43, no. 2, pp. 429–449, 2014.

[95] S. Li, W. Zhang, and A. B. Chan, “Maximum-margin structured learning with deep networks for 3d human pose estimation,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 2848–2856.

[96] T. Ogita, S. Rump, and S. Oishi, “Accurate sum and dot product with applications,” in *2004 IEEE International Conference on Robotics and Automation*, 2004, pp. 152–155.

[97] J. Simon, *XSum*, MATLAB Central File Exchange, Jan. 2021.

[98] W. Kahan, “Pracniques: Further remarks on reducing truncation errors,” *Communications of the ACM*, vol. 8, no. 1, p. 40, Jan. 1965.

[99] I. Babŭska, “Numerical stability in mathematical analysis,” *Information Processing*, vol. 68, pp. 11–23, 1969.

[100] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[101] P. J. Huber, “Robust Estimation of a Location Parameter,” *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[102] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[103] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[104] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 675–678.

[105] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The Field D* algorithm," *Journal of Field Robotics*, vol. 23, pp. 79–101, 2006.

[106] D. Ferguson and A. Stentz, "The Field D* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-19, Jun. 2005.

[107] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," in *Proceedings of 12th International Symposium on Robotics Research (ISRR '05)*, Oct. 2005, pp. 239–253.

[108] S. Koenig and M. Likhachev, "D*Lite," in *Eighteenth National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 476–483.

[109] S. Koenig and M. Likhachev, "Incremental A*," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14, MIT Press, 2002.

[110] A. Stentz, "The focussed D* algorithm for real-time replanning," in *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1652–1659.

[111] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," *International Journal of Robotics and Automation*, vol. 10, pp. 89–100, 1993.

[112] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[113] L. R. Ford, "Network flow theory.," RAND Corporation, Santa Monica, CA, Tech. Rep. RAND-P-923, 1956.

[114] R. Bellman, "On a routing problem," RAND Corporation, Santa Monica, CA, Tech. Rep. RAND-P-1000, 1956.

[115] A. Kosson, V. Chiley, A. Venigalla, J. Hestness, and U. Koster, "Pipelined backpropagation at scale: Training large models without batches," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 479–501.

[116] Khronos OpenCL Working Group, *The opencl specification, version 2.0*, ed. by L. Howes and A. Munshi, 2015.

[117] U. Meyer and P. Sanders, "Δ-stepping: A parallel single source shortest path algorithm," in *Proceedings of the 6th Annual European Symposium on Algorithms*, ser. ESA '98, 1998, pp. 393–404.

[118] U. Meyer and P. Sanders, "Δ-stepping: A parallelizable shortest path algorithm," *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, 2003, 1998 European Symposium on Algorithms.

[119] E. Duriakova, D. Ajwani, and N. Hurley, "Engineering a parallel Δ-stepping algorithm," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 609–616.

[120] V. T. Chakaravarthy, F. Checconi, F. Petrini, and Y. Sabharwal, "Scalable single source shortest path algorithms for massively parallel systems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 889–901.

[121] G. E. Blelloch, Y. Gu, Y. Sun, and K. Tangwongsan, "Parallel shortest paths using radius stepping," in *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '16, Pacific Grove, California, USA: Association for Computing Machinery, 2016, pp. 443–454.

[122] Q. Peng, Y. Yu, and W. Wei, "The shortest path parallel algorithm on single source weighted multi-level graph," in *2009 Second International Workshop on Computer Science and Engineering*, vol. 2, 2009, pp. 308–311.

[123] A. Prasad, S. K. Krishnamurthy, and Y. Kim, "Acceleration of Dijkstra's algorithm on multi-core processors," in *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, 2018, pp. 1–5.

[124] M. Nazarifard and D. Bahrepour, "Efficient implementation of the Bellman-Ford algorithm on GPU," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, 2017, pp. 0773–0778.

[125] F. Busato and N. Bombieri, "An efficient implementation of the bellman-ford algorithm for Kepler GPU architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2222–2233, 2016.

[126] G. G. Surve and M. A. Shah, "Parallel implementation of Bellman-Ford algorithm using CUDA architecture," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 2, 2017, pp. 16–22.

[127] A. Davidson, S. Baxter, M. Garland, and J. D. Owens, "Work-efficient parallel GPU methods for single-source shortest paths," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 349–359.

[128] S. Kumar, A. Misra, and R. S. Tomar, "A modified parallel approach to single source shortest path problem for massively dense graphs using CUDA," in *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*, 2011, pp. 635–639.

[129] P. Harish and P. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," vol. 4873, Dec. 2007, pp. 197–208.

[130] B. O. Community, *Blender - a 3D modelling and rendering package*, Stichting Blender Foundation, Amsterdam: Blender Foundation, 2018.

[131] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, `https://ompl.kavrakilab.org`.

[132] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[133] S. Marcel and Y. Rodriguez, "Torchvision: The machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10, Firenze, Italy: Association for Computing Machinery, 2010, pp. 1485–1488.

[134] D. Ulyanov, *Multicore-TSNE*, `https://github.com/DmitryUlyanov/Multicore-TSNE`, 2016.

[135] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[136] L. van der Maaten, "Accelerating t-SNE using tree-based algorithms," *Journal of Machine Learning Research*, vol. 15, no. 93, pp. 3221–3245, 2014.

[137] A. H. Jiang, D. L. Wong, G. Zhou, D. G. Andersen, J. Dean, G. R. Ganger, G. Joshi, M. Kaminsky, M. Kozuch, Z. C. Lipton, and P. Pillai, "Accelerating deep learning by focusing on the biggest losers," *CoRR*, vol. abs/1910.00762, 2019.

[138] Q. Dong, S. Gong, and X. Zhu, "Class rectification hard mining for imbalanced deep learning," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1869–1878.

[139] C. Zhang, C. Öztireli, S. Mandt, and G. Salvi, "Active mini-batch sampling using repulsive point processes," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19, Honolulu, Hawaii, USA: AAAI Press, 2019.

[140] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," *CoRR*, vol. abs/1803.00942, 2018.

[141] T. B. Johnson and C. Guestrin, "Training deep models faster with robust, approximate importance sampling," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 7276–7286.

[142] M. Baines, S. Bhosale, V. Caggiano, N. Goyal, S. Goyal, M. Ott, B. Lefaudeux, V. Liptchinsky, M. Rabbat, S. Sheiffer, A. Sridhar, and M. Xu, *FairScale: A general purpose modular PyTorch library for high performance and large scale training*, `https://github.com/facebookresearch/fairscale`, 2021.

[143] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, *Zero: Memory optimizations toward training trillion parameter models*, ArXiv, Oct. 2019.

[144] Y. Xu, H. Lee, D. Chen, H. Choi, B. A. Hechtman, and S. Wang, "Automatic cross-replica sharding of weight update in data-parallel training," *CoRR*, vol. abs/2004.13336, 2020.

[145] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? The INRIA aerial image labeling benchmark," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, IEEE, 2017.

[146] *Atlas: The Louisiana statewide GIS*, http://atlas.lsu.edu, Baton Rouge, Louisiana, USA, 2021.

[147] J. Yuan, S. S. Gleason, and A. M. Cheriyadat, "Systematic benchmarking of aerial image segmentation," *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 6, pp. 1527–1531, 2013.

[148] V. Iglovikov, S. Mushinskiy, and V. Osin, "Satellite imagery feature detection using deep convolutional neural network: A kaggle competition," *CoRR*, vol. abs/1706.06169, 2017.

[149] OpenStreetMap contributors, *Planet dump retrieved from https://planet.osm.org*, `https://www.openstreetmap.org`, 2017.

[150] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu, "Neural SLAM," *CoRR*, vol. abs/1706.09520, 2017.

[151] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014.

[152] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.