Algorithms for Robust Autonomous Navigation in Human Environments

by

Michael F. Everett

S.B., Massachusetts Institute of Technology (2015)

S.M., Massachusetts Institute of Technology (2017)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2020

© Massachusetts Institute of Technology 2020. All rights reserved.

Author .....
Department of Mechanical Engineering
June 22, 2020

Certified by .....
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Alberto Rodriguez
Associate Professor of Mechanical Engineering
Thesis Supervisor

Certified by .....
John Leonard
Samuel C. Collins Professor of Mechanical and Ocean Engineering
Thesis Committee Chair

Accepted by .....
Nicolas Hadjiconstantinou
Graduate Officer, Department of Mechanical Engineering

Algorithms for Robust Autonomous Navigation in Human Environments

by

Michael F. Everett

Submitted to the Department of Mechanical Engineering
on June 22, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Today's robots are designed *for* humans, but are rarely deployed *among* humans. This thesis addresses problems of perception, planning, and safety that arise when deploying a mobile robot in human environments. A first key challenge is that of quickly navigating to a human-specified goal – one with known semantic type, but unknown coordinate – in a previously unseen world. This thesis formulates the contextual scene understanding problem as an image translation problem, by learning to estimate the planning cost-to-go from aerial images of similar environments. The proposed perception algorithm is united with a motion planner to reduce the amount of exploration time before finding the goal. In dynamic human environments, pedestrians also present several important technical challenges for the motion planning system. This thesis contributes a deep reinforcement learning-based (RL) formulation of the multiagent collision avoidance problem, with relaxed assumptions on the behavior model and number of agents in the environment. Benefits include strong performance among many nearby agents and the ability to accomplish long-term autonomy in pedestrian-rich environments. These and many other state-of-the-art robotics systems rely on Deep Neural Networks for perception and planning. However, blindly applying deep learning in safety-critical domains, such as those involving humans, remains dangerous without formal guarantees on robustness. For example, small perturbations to sensor inputs are often enough to change network-based decisions. This thesis contributes an RL framework that is certified robust to uncertainties in the observation space.

Thesis Supervisor: Jonathan P. How
Title: R. C. Maclaurin Professor of Aeronautics and Astronautics

Thesis Supervisor: Alberto Rodriguez
Title: Associate Professor of Mechanical Engineering

Thesis Committee Chair: John Leonard
Title: Samuel C. Collins Professor of Mechanical and Ocean Engineering

Acknowledgments

I did not plan to be at MIT for so long and definitely did not plan to do a PhD. There are a lot of people to thank for making it a great experience.

First, I want to thank my advisor, Professor Jonathan How, for his support, motivation, and mentorship throughout graduate school. He sets the bar high and invests a ton of his own time to help his students be successful.

Thank you to the thesis committee: Professor John Leonard and Professor Alberto Rodriguez have helped me to think critically about where my work fits into the field and how it ties together, and they have also given extensive, invaluable advice about how to begin my career after the PhD.

The Aerospace Controls Laboratory (ACL) has been an awesome place to learn to become a researcher. Although I study neither Aerospace nor Controls, the wide range of people and projects in the lab have helped me learn a little about so many different topics. Thank you in particular to Drs. Steven Chen, Justin Miller, Brett Lopez, Shayegan Omidshafiei, and Kasra Khosoussi who taught and inspired me more than they realize. The pursuit of international cuisine while traveling with Björn Lütjens and Samir Wadhwania (inspired by Dong-Ki Kim) was also quite memorable. Dr. Shih-Yuan Liu and Parker Lusk have been guiding voices about writing code the right way, even if it is not the fastest way that day (along with other great advice). Thank you to Building 41 for providing some perspective on Building 31.

Thank you to Ford Motor Company, and in particular, ACL alumnus Dr. Justin Miller and Dr. Jianbo Lu: they have supported my research through thoughtful discussions during monthly telecons and campus visits over the years, as well as financially. It has been motivating to have a group of collaborators that are also working hard to see these technologies come to life.

Thank you to Meghan Torrence for helping me realize I should do a PhD and for making it a lot more fun.

Finally, I thank my family members for their support: Mom ('79 SB, '81 SM), Dad ('76 SB, '91 PhD), Tim, Katie ('12 SB, '13 MEng), and Patrick ('17 SB).

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms

A3C Asynchronous Advantage Actor-Critic. 66, 67

BFS Breadth-First Search. 48

CADRL Collision Avoidance with Deep Reinforcement Learning. 11, 16, 17, 58, 64–66, 70, 73, 74, 77, 81, 87, 88

CARRL Certified Adversarial Robustness for Deep Reinforcement Learning. 114, 116–118, 120–131

CNN Convolutional Neural Network. 67

CROWN Algorithm from [132]. 131

DC2G Deep Cost-to-Go. 11, 39, 40, 46–48, 51–55, 133

DNN Deep Neural Networks. 101

DNN Deep Neural Network. 26, 28, 35, 63–66, 70, 73, 76, 101, 104, 106, 107, 110, 112, 113, 115, 116, 128, 132, 135, 137, 139, 140, 142

DQN Deep Q-Network. 12, 49, 104, 106, 112, 113, 122–126, 130

DRLMACA Deep Reinforcement Learning Multiagent Collision Avoidance. 11, 81, 85–87

FGST Fast Gradient Sign method with Targeting. 118, 124

FOV Field of View. 30, 46

GA3C Hybrid GPU/CPU Asynchronous Advantage Actor-Critic. 16, 66, 70, 79

GA3C-CADRL Hybrid GPU/CPU Asynchronous Advantage Actor-Critic (GA3C) CADRL. 11, 12, 16, 70, 73–87, 90, 91, 94, 99, 122, 133

GA3C-CADRL-10 GA3C-CADRL trained on ≤ 10 agents. 16, 81, 85

GA3C-CADRL-10-LSTM GA3C-CADRL-10 with an LSTM architecture. 77, 79, 82–85, 87, 88, 90, 92, 94

GA3C-CADRL-10-WS-4 GA3C-CADRL-10 with WS architecture with 4-agent capacity. 79, 85

GA3C-CADRL-10-WS-6 GA3C-CADRL-10 with WS architecture with 6-agent capacity. 79

GA3C-CADRL-10-WS-8 GA3C-CADRL-10 with WS architecture with 6-agent capacity. 79

GA3C-CADRL-4 GA3C-CADRL trained on ≤ 4 agents. 16, 81, 82, 85

GA3C-CADRL-4-LSTM GA3C-CADRL-4 with an LSTM architecture. 79, 85

GA3C-CADRL-4-WS-4 GA3C-CADRL-4 with 4-agent capacity. 79, 83, 85

GA3C-CADRL-4-WS-6 GA3C-CADRL-4 with 6-agent capacity. 79, 85

GA3C-CADRL-4-WS-8 GA3C-CADRL-4 with 6-agent capacity. 79

IPL Inverse Path Length. 47

LP Linear Programming. 12, 107, 108, 116, 128, 129

LSTM Long Short-Term Memory. 11, 58, 59, 70–73, 77, 79, 80, 83, 85, 94–99, 134

MPC Model Predictive Control. 60, 135

ORCA Optimal Reciprocal Collision Avoidance. 79, 81–84, 87, 122, 127

ReLU Rectified Linear Unit. 107, 110–112, 116

RGB-D Color (Red, Green, Blue) & Depth. 31, 36, 38

RL Reinforcement Learning. 11, 23, 24, 27, 28, 35, 49, 51, 52, 58, 62, 66–70, 73, 74, 76, 77, 81, 82, 85, 99, 101–108, 110, 112, 113, 120, 122, 126, 128, 129, 132–135

ROS Robot Operating System. 59

SA-CADRL Socially Aware CADRL. 73, 74, 77–79, 81–85, 87

SLAM Simultaneous Localization and Mapping. 35, 46

SMT Satisfiability Modulo Theory. 107

SPL Success Weighted by Inverse Path Length. 47, 52

WS weights shared. 16, 83, 85

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

1.1 Overview

Today's robots are designed for humans. Drone videography [1] and light shows [2] consist of human-entertaining robots; modern warehouses [3] and disaster relief scenarios [4] consist of human-supporting robots; hospital wards [5, 6] and vehicle test tracks [7, 8] consist of human-saving robots. Although these robots are built *for* humans, they are rarely deployed *among* humans. This thesis addresses several key technical challenges that prevent robots from being deployed in human environments.

In particular, this thesis focuses on the decision-making process to enable autonomous navigation: choosing actions that guide a robot toward a goal state as quickly as possible. This problem of robot navigation has been the focus of decades of robotics research [9], but the assumptions that enable those capabilities also constrain the types of environments they can operate in. Existing algorithms often make unrealistic assumptions about human environments, which is why robots still have a limited presence outside of research labs and factories. While human environments provide a common framework to motivate these challenges, the algorithms provided by this thesis address core issues that span the space of perception, planning, and safety in robotics and could improve future robot operation in generic environments.

The underlying idea in this thesis is that although many aspects of the world are difficult for an engineer to model and design algorithms for, humans can design

algorithms that, by observing data about the world, teach other algorithms (running on robots) to make intelligent decisions. This concept is the guiding principle behind learning-based methods. Recent advances in computation devices [10], the backpropagation algorithm [11], and the explosion of available data [12] has produced great interest in the topic of *deep learning* [13].

Still, the connection of robotics and deep learning raises challenges that are not present in generic learning tasks. For example, while many learning tasks (e.g., image classification: what type of object is in this picture?) can leverage huge, curated datasets, robotics data is often sparse, as meaningful data can be expensive or dangerous to collect. Thus, this thesis describes how to formulate the problems such that existing datasets can be re-purposed for robotics or, if static datasets are insufficient, simulations can provide a proxy for real data. More importantly, engineers have a responsibility to understand the safety implications of the robots they produce [14] – this remains a fundamental issue at the intersection of robotics and learning. With these challenges in mind, this thesis provides algorithms for robust autonomous navigation in human environments.

The following sections motivate and introduce the problems addressed by this thesis (Section 1.2), describe the technical contributions (Section 1.3), and thesis structure (Section 1.4).

1.2 Problem Statement

This thesis develops learning-based algorithms to address key challenges in deploying mobile robots in human environments. The following questions are investigated: (i) How to guide motion plans with scene context when the goal is only described semantically, and there is no prior map? (ii) How to navigate safely among pedestrians, of which there may be a large, possibly time-varying number? (iii) How to use learned policies under adversarial observation uncertainty? The following sections provide motivation and elaborate on these questions, while the technical problem formulations are left for the subsequent chapters.

1.2.1 Planning Without a Known Goal Coordinate or Prior Map

Robot planning algorithms traditionally assume access to a map of the environment ahead of time. However, it is intractable to collect and maintain a detailed prior map of every human environment (e.g., inside everyone's house). Moreover, a goal coordinate is rather ungrounded without a map or external sensing, such as GPS. Humans more naturally describe what they are looking for or where they are going by *semantics* (e.g., "I'm looking for *my keys* in the *living room*", rather than "I left my keys at [0.24, 5.22]"). Thus, this work assumes that a robot has never visited its particular environment before, its goal is only described semantically, and the objective is to reach the goal as quickly as possible.

1.2.2 Planning Among Dynamic, Decision-Making Obstacles, such as Pedestrians

Pedestrians are an example of dynamic/moving obstacles, but few robot motion planning algorithms account for the fact that pedestrians also are decision-making agents, much like the robot. Reasoning about interaction (e.g., if I do this, you will do that) is crucial for seamless robot navigation among pedestrians, but was previously too computationally intensive to be computed within the real-time decision-making process. In this problem, the robot's planning objective is to reach the goal position as quickly as possible without colliding with the large, possibly time-varying number of dynamic, decision-making obstacles.

1.2.3 Deep RL with Adversarial Sensor Uncertainty

The algorithmic approach (deep learning) taken to address the previous challenges is highly sensitive to small perturbations in measurements. Unfortunately, real-world robotic sensors are subject to imperfections/noise, raising a fundamental issue in applying the algorithms to real systems. To bridge this gap, the objective of this

work is to modify a learned policy to maximize performance when observations have been perturbed by an adversarial (worst-case) noise process.

1.3 Technical Contributions and Thesis Structure

1.3.1 Contribution 1: Planning Beyond the Sensing Horizon Using a Learned Context

This contribution [15] addresses the problem of quickly navigating in a previously unseen world, to a goal with known semantic type, but unknown coordinate. The key idea is a novel formulation of contextual scene understanding as an image translation problem, which unlocks the use of powerful supervised learning techniques [16]. In particular, an algorithm for learning to estimate the planning cost-to-go is proposed and trained with supervision from Dijkstra's algorithm [17] and a dataset of satellite maps from areas with similar structure to the test environments. The benefit of this contribution is the ability to quickly reach an unknown goal position in previously unseen environments.

1.3.2 Contribution 2: Collision Avoidance in Pedestrian-Rich Environments with Deep Reinforcement Learning

This contribution [18] addresses the problem of motion planning in pedestrian-rich environments. Building on our prior work [19, 20], the problem is formulated as a partially observable, decentralized planning problem, solved with deep reinforcement learning. The key technical innovations are in algorithmic improvements that relax the need for assumptions on the behavior model and numbers of agents in the environment. The main benefits are in increased performance as the number of agents in the environment grows, and the ability to accomplish long-term autonomy in pedestrian-rich environments.

1.3.3 Contribution 3: Certified Adversarial Robustness for Deep Reinforcement Learning

This contribution [21] focuses on the challenges of using deep RL in safety-critical domains. In particular, uncertainty in sensor measurements (due to noise or adversarial perturbations) is considered formally for the first time in the deep RL literature. The key idea is to account for the potential worst-case outcome of a candidate action, by propagating sensor uncertainty through a deep neural network. A robust optimization formulation is solved efficiently by extending recent neural network robustness analysis tools [22]. By also providing a certificate of solution quality, the framework enables *certifiably robust* deep RL for deployment in the presence of real-world uncertainties.

1.3.4 Contribution 4: Demonstrations in Simulation & on Multiple Robotic Platforms

The goal of the thesis is to provide algorithmic innovations to enable robot navigation in human environments. Thus, simulation and hardware experiments are designed to test the algorithmic assumptions and evaluate the proposed approaches under realistic or real operating conditions.

1.4 Thesis Structure

The rest of the thesis is structured as follows:

- Chapter 2 describes background material, including relevant machine learning concepts.
- Chapter 3 presents the framework for planning beyond the sensing horizon using a learned context (Contribution 1) with low- and high-fidelity simulation experiments and validation of the approach on real data collected on a mobile robot (Contribution 4).

  The content of this chapter is based on: Michael Everett, Justin Miller, and Jonathan P. How. "Planning Beyond The Sensing Horizon Using a Learned

Context". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China, 2019. URL: https://arxiv.org/pdf/1908.09171.pdf.

- Chapter 4 presents the multiagent collision avoidance framework (Contribution 2) and shows extensive simulation and hardware experiments (Contribution 4). The content of this chapter is based on: Michael Everett, Yu Fan Chen, and Jonathan P. How. "Collision Avoidance in Pedestrian-Rich Environments with Deep Reinforcement Learning". In: *International Journal of Robotics Research (IJRR)* (2019), in review. URL: https://arxiv.org/pdf/1910.11689.pdf.
- Chapter 5 presents the certified adversarial robustness framework for deep RL (Contribution 3), and demonstrates the algorithmic advantages in two simulation tasks and under different types of adversarial activity (Contribution 4). The content of this chapter is based on: Michael Everett*, Björn Lütjens*, and Jonathan P. How. "Certified Adversarial Robustness in Deep Reinforcement Learning". In: *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* (2020), in review. URL: https://arxiv.org/pdf/2004.06496.pdf.
- Chapter 6 summarizes the thesis contributions and describes future research directions.

A literature review is provided for each of the sub-topics in Chapters 3 to 5.

Chapter 2

Preliminaries

In this chapter, we introduce some important topics in machine learning for background.

The learning problems considered in this thesis are instances of supervised learning and reinforcement learning. A reference for deep learning fundamentals is [13]. An excellent description of the fundamentals of reinforcement learning can be found online [23] or in a detailed textbook [24]. This chapter summarizes some of the key ideas that form the foundation of later chapters.

2.1 Supervised Learning

In supervised learning, there is a static dataset, $\mathcal{D}$, of N pairs of features, $\mathbf{x}_i$, and labels, $\mathbf{y}_i$: $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$. For example, each feature could be a picture, and each label could be what type of object is in that picture. The objective is typically to use the "training data", $\mathcal{D}$, to learn a model $f : X \to Y$ to predict the labels $\mathbf{y}$ associated with previously unseen "test data" (i.e., inputs $\mathbf{x}_i \notin \mathcal{D}$). This model is parameterized by weight vector, θ, and is written as $\hat{\mathbf{y}} = f(\mathbf{x}; \theta)$, where the $\hat{}$ denotes that $\hat{\mathbf{y}}$ is merely an estimate of the true label, $\mathbf{y}$. The weights control how the model represents the relationship between the features and labels. To quantify how well the model captures the relationship, one can define a loss function, $\mathcal{L}$, to convey how closely the predicted labels, $\hat{\mathbf{y}}$, match the true labels, $\mathbf{y}$ (e.g., the mean-

square-error loss, $\mathcal{L}_{MSE} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} ||\mathbf{y}_i - f(\mathbf{x}_i; \theta)||_2^2)$. The model parameters, θ, can be adjusted to minimize this loss function, often through gradient descent-like optimization algorithms. While this formulation is straightforward to define, practical challenges include acquiring enough useful data, choosing a loss function to describe performance, and developing efficient/effective optimization algorithms to make learning (adjusting the parameters) to a desired level of performance occur in a reasonable amount of time.

2.1.1 Deep Learning

With real-world data, the relationship between features and labels is often quite complicated. For example, describing what colors, shapes, and components describe a car in an image quickly becomes intractable. Instead, deep learning-based methods use a particular class of models, Deep Neural Network (DNN) to represent f. The model parameters, θ, usually represent the DNN layers' weights and biases, and could include the overall model architecture, since design decisions like the number of layers and layer size often have a big impact on model performance.

2.2 Reinforcement Learning

Supervised learning describes problems in which a model makes a decision based on the current input, and that decision's impact is felt immediately. However, many robotics problems are better formulated as making a decision that could have impact on a system that evolves for some time into the future. For instance, a supervised learning problem for a robot is to identify pedestrians' locations in the image returned by the camera, but the robot's decision of where to drive next should reason about how the robot and pedestrians' decisions might influence each other over the next several seconds, which is better described by a Markov Decision Process.

2.2.1 Markov Decision Processes

Many robotics problems can be formulated as Markov Decision Processes (MDPs) [25], so long as all information needed about the system's history can be described in a fixed-size vector at the current timestep. An MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, such that at each discrete timestep:

- $\mathbf{s} \in \mathcal{S}$ is the current system state,
- $\mathbf{a} \in \mathcal{A}$ is the action taken by following the policy $\pi(\mathbf{a}|\mathbf{s})$,
- $\mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the distribution of next states, s', given that the action $\mathbf{a}$ is taken from the current state $\mathbf{s}$,
- $r = \mathcal{R}(\mathbf{s}, \mathbf{s}', \mathbf{a})$ provides a scalar reward for taking action $\mathbf{a}$ in state $\mathbf{s}$ and reaching next state $\mathbf{s}'$,
- $R = \sum_{t=0}^{\infty} \gamma^t r_t$ is the return, which is an accumulation of all future rewards discounted by $\gamma \in [0, 1)$.

A value function numerically describes how "good" it is for the system to be in its current state, by describing the expected return that would be received by following a policy π starting from the current state or state-action pair:

$$V^{\pi}(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}_0 = \mathbf{s}] \qquad \text{State Value Function} \tag{2.1}$$

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}[R|\mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}] \qquad \text{State} - \text{Action Value Function,} \tag{2.2}$$

where Eq. (2.2) is often called the Q-function. Note that the policy π is used to select actions for all future timesteps (with the exception of $\mathbf{a}_0$ in Eq. (2.2)). The optimal policy π^* is traditionally defined as one which maximizes $V^{\pi}(\mathbf{s})$ or $Q^{\pi}(\mathbf{s}, \mathbf{a})$, and following π^* produces value functions written as $V^*(\mathbf{s})$ or $Q^*(\mathbf{s}, \mathbf{a})$, respectively.

In this thesis, the action is sometimes written as $\mathbf{u}$ to match the control literature.

2.2.2 Learning a Policy

Although there are many techniques for computing a policy or value function when the MDP is known, obtaining the system's transition function is often difficult. Re-

inforcement Learning (RL) addresses this issue by assuming $\mathcal{T}, \mathcal{R}$ are unknown, and the environment simply provides a reward as feedback at each timestep. The premise in RL is that a dataset of $(\mathbf{s}, \mathbf{a}, r)$ tuples can be used to develop a good policy.

Model-based RL methods obtain the policy by using $(\mathbf{s}, \mathbf{a}, r)$ to approximate $\mathcal{T}, \mathcal{R}$, or other functions that describe the future (e.g., a robot's probability of collision with an obstacle), then use *planning* algorithms to compute a policy from the learned system model. Model-free RL methods directly learn to estimate the value function, state-value function, policy, or some combination or variant of those.

2.2.3 Deep Reinforcement Learning

Because $\mathcal{T}$ can be quite complicated and the policy space can be gigantic in real-world systems, both model-based and model-free methods can benefit from approximating the optimal model (dynamics, value, policy, etc.) with a model from a very expressive model class, such as DNNs. Deep RL is just RL with a DNN to represent some of the learned models.

2.3 Summary

This chapter briefly described the problems of supervised learning and reinforcement learning, which lay the foundations of the subsequent chapters.

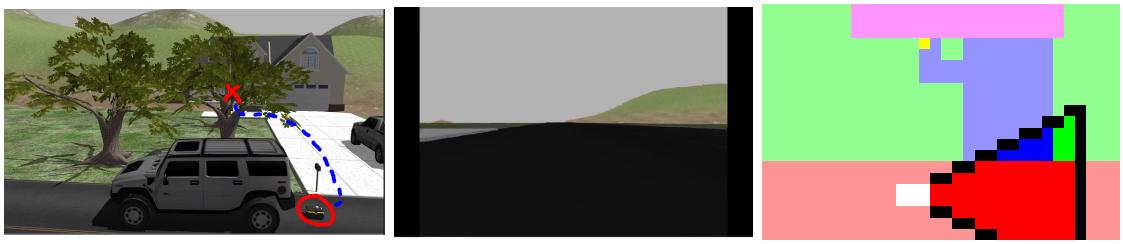
Chapter 3

Planning Beyond the Sensing Horizon Using a Learned Context

3.1 Introduction

A key topic in robotics is the use of automated robotic vehicles for last-mile delivery. A standard approach is to visit and map delivery environments ahead of time, which enables the use of planning algorithms that guide the robot toward a specific goal coordinate in the map. However, the economic inefficiency of collecting and maintaining maps, the privacy concerns of storing maps of people's houses, and the challenges of scalability across a city-wide delivery system are each important drawbacks of the pre-mapping approach. This motivates the use of a planning framework that does not need a prior map. In order to be a viable alternative framework, the time required for the robot to locate and reach its destination must remain close to that of a prior-map-based approach.

Consider a robot delivering a package to a new house's front door (Fig. 3-1). Many existing approaches require delivery destinations to be specified in a format useful to the robot (e.g., position coordinates, heading/range estimates, target image), but collecting this data for every destination presents the same limitations as prior mapping. Therefore the destination should be a high-level concept, like "go to the front door." Such a destination is intuitive for a human, but without actual co-



(a) Oracle's view (b) Robot's view (c) Semantic Map

Figure 3-1: Robot delivers package to front door. If the robot has no prior map and does not know where the door is, it must quickly search for its destination. Context from the onboard camera view (b) can be extracted into a lower-dimensional semantic map (c), where the white robot can see terrain within its black FOV.

ordinates, difficult to translate into a planning objective for a robot. The destination will often be beyond the robot's economically-viable sensors' limited range and field of view. Therefore, the robot must explore [26, 27] to find the destination; however, pure exploration is slow because time is spent exploring areas unlikely to contain the goal. Therefore, this paper investigates the problem of efficiently planning beyond the robot's line-of-sight by utilizing *context* within the local vicinity. Existing approaches use context and background knowledge to infer a geometric understanding of the high-level goal's location, but the representation of background knowledge is either difficult to plan from [28–36], or maps directly from camera image to action [37–44], reducing transferability to real environments.

This work proposes a solution to efficiently utilize context for planning. Scene context is represented in a semantic map, then a learned algorithm converts context into a search heuristic that directs a planner toward promising regions in the map. The context utilization problem (determination of promising regions to visit) is uniquely formulated as an image-to-image translation task, and solved with U-Net/GAN architectures [16, 45] recently shown to be useful for geometric context extraction [46]. By learning with semantic gridmap inputs instead of camera images, the planner proposed in this work could be more easily transferred to the real world without the need for training in a photo-realistic simulator. Moreover, a standard local collision avoidance algorithm can operate in conjunction with this work's global planning algorithm, making the framework easily extendable to environments with

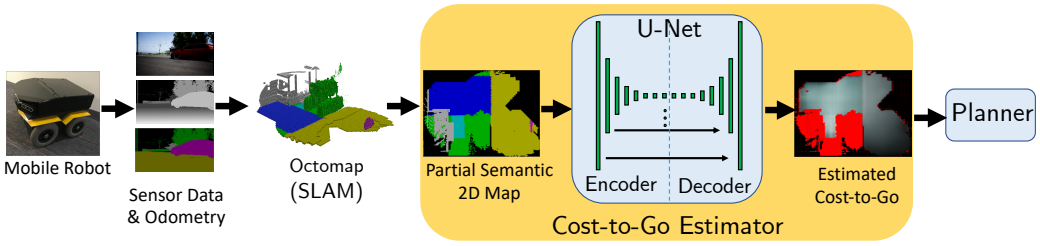


Figure 3-2: System architecture. To plan a path to an unknown goal position beyond the sensing horizon, a robot's sensor data is used to build a semantically-colored gridmap. The gridmap is fed into a U-Net to estimate the cost-to-go to reach the unknown goal position. The cost-to-go estimator network is trained offline with annotated satellite images. During online execution, the map produced by a mobile robot's forward-facing camera is fed into the trained network, and cost-to-go estimates inform a planner of promising regions to explore.

dynamic obstacles.

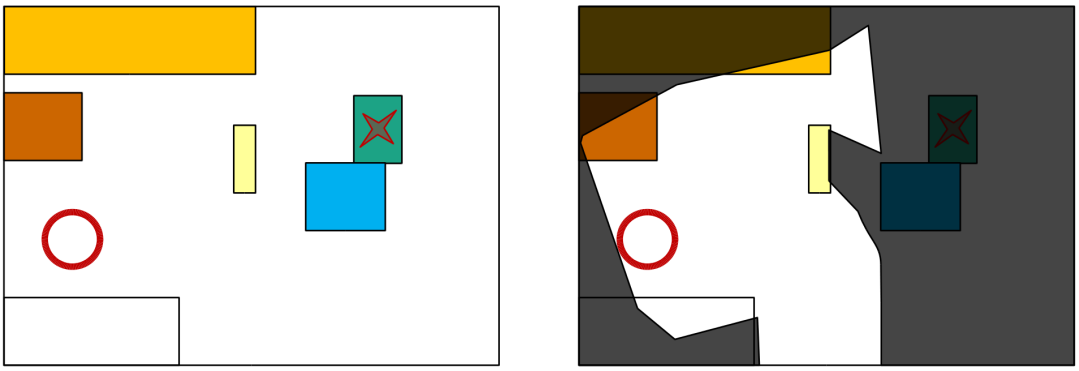
The contributions of this work are (i) a novel formulation of utilizing context for planning as an image-to-image translation problem, which converts the abstract idea of scene context into a planning metric, (ii) an algorithm to efficiently train a cost-to-go estimator on typical, partial semantic maps, which enables a robot to learn from a static dataset instead of a time-consuming simulator, (iii) demonstration of a robot reaching its goal 189% faster than a context-unaware algorithm in simulated environments, with layouts from a dataset of real last-mile delivery domains, and (iv) an implementation of the algorithm on a vehicle with a forward-facing RGB-D + segmentation camera in a high-fidelity simulation. Software for this work is published at `https://github.com/mit-acl/dc2g`.

3.2 Background

This section formally defines the problem statement solved in this chapter and describes several bodies of work that solve related problems.

3.2.1 Problem Statement

Let an environment be described by a 2D grid of size (h, w), where each grid cell is occupied by one of n_c semantic classes (types of objects). The 2D grid is called the



(a) Semantic Gridmap, S (b) Partial Semantic Gridmap, $S_{partial}$

Figure 3-3: Problem Statement Visualization. In (a), a red circle indicates an agent inside a semantic gridmap with objects of various semantic class (colored rectangles). The red star denotes the goal region, but the agent is only provided with the goal object class (green object). Furthermore, the agent is not provided with the full environment map a priori – the dark area in (b) depicts regions of the semantic map that have not been observed by the agent yet. As the agent explores the environment, it can use onboard sensors to uncover more of the map to help find the green object.

semantic gridmap, $S \in \{1, \ldots, n_c\}^{h \times w}$. Let the *goal object* be one of the semantic classes, $g_c \in \{1, \ldots, n_c\}$; the cells in the map of class g_c is called the *goal region*, $\mathcal{G} = \{(p_x, p_y) \in \{1, \ldots, w\} \times \{1, \ldots, h\} | S_{p_x, p_y} = g_c\}$.

To model the fact that human-built environments are different yet have certain structures in common, let there be a semantic map-generating distribution, S. We populate a training set, $\mathcal{S}_{train}$, of maps by taking N i.i.d. samples from S, such that $\mathcal{S}_{train} = \{S_1, \ldots, S_N\} \sim \mathsf{S}$.

An agent is randomly placed at position $\mathbf{p}_0 = (p_{x,0}, p_{y,0})$ in a previously unseen test map, S. The agent does not have access to the full environment map S; at each timestep t, the agent can create and use a partial semantic map, $S_{partial,t}$, using limited field-of-view/range sensor observations from timesteps $0 \leq 0 \leq t$. The agent knows the goal's semantic class, g_c, but not the goal region, $\mathcal{G}$. Thus, the observable state $\mathbf{s}_t^o = [p_{x,t}, p_{y,t}, S_{partial,t}, g_c]$. A policy, $\pi : (\mathbf{s}_t) \mapsto \mathbf{p}_{t+\Delta t}$, is developed with the

objective of moving the agent to the goal region as quickly as possible,

$$\underset{\pi}{\text{argmin}} \quad \mathbb{E}\left[t_g | \mathbf{s}_t,\, \pi, S\right] \tag{3.1}$$

$$s.t. \quad \mathbf{p}_{t+\Delta t} = \pi(\mathbf{s}_t^o) \tag{3.2}$$

$$\mathbf{p}_{t_g} \in \mathcal{G} \tag{3.3}$$

$$S, \underbrace{S_1, \ldots, S_N}_{\mathcal{S}_{train}} \sim \mathsf{S}. \tag{3.4}$$

This definition formalizes the "ObjectGoal" task defined in [47]. In our work, the agent is not allowed any prior exploration/map of the test environments, but can use the training set to extract properties of the map-generating distribution. The technical problem statement can be visualized in Fig. 3-3.

3.2.2 Related Work

Planning & Exploration

Classical planning algorithms rely on knowledge of the goal coordinates (A*, RRT) and/or a prior map (PRMs, potential fields), which are both unavailable in this problem. Receding-horizon algorithms are inefficient without an accurate heuristic at the horizon, typically computed with goal knowledge. Rather than planning to a destination, the related field of exploration provides a conservative search strategy. The seminal frontier exploration work of Yamauchi provides an algorithm that always plans toward a point that expands the map [26]. Pure exploration algorithms often estimate *information gain* of planning options using geometric context, such as the work by Stachniss et al. [27]. However, exploration and search objectives differ, meaning the exploration robot will spend time gaining information in places that are useless for the search task.

Context for Object Search

Leveraging scene context is therefore fundamental to enable object search that outperforms pure exploration. Many papers consider a single form of context. Geometric context (represented in occupancy grids) is used in [48–51], but these works also assume knowledge of the goal location for planning. Works that address true object search usually consider semantic object relationships as a form of context instead. Joho et al. showed that decision trees and maximum entropy models can be trained on object-based relationships, like positions and attributes of common items in grocery stores [28]. Because object-based methods require substantial domain-specific background knowledge, Samadi et al. and Kollar et al. automate the background data collection process by using Internet searches [29, 30]. Object-based approaches have also noted that the spatial relationships between objects are particularly beneficial for search (e.g., keyboards are often *on* desks) [31–33], but are not well-suited to represent geometries like floorplans/terrains. Hierarchical planners improve search performance via a human-like ability to make assumptions about object layouts [35, 36].

To summarize, existing uses of context either focus on relationships between objects or the environment's geometry. These approaches are too specific to represent the combination of various forms of context that are often needed to plan efficiently.

Deep Learning for Object Search

Recent works use deep learning to represent scene context. Several approaches consider navigation toward a semantic concept (e.g., go to the kitchen) using only a forward-facing camera. These algorithms are usually trained end-to-end (image-to-action) by supervised learning of expert trajectories [37, 39, 52] or reinforcement learning in simulators [40–44]. Training such a general input-output relationship is challenging; therefore, some works divide the learning architecture into a deep neural network for each sub-task (e.g., mapping, control, planning) [39, 42, 43].

Still, the format of context in existing, deep learning-based approaches is too

general. The difficulty in learning how to extract, represent, and use context in a generic architecture leads to massive computational resource and time requirements for training. In this work, we reduce the dimensionality (and therefore training time) of the learning problem by first leveraging existing algorithms (semantic SLAM, image segmentation) to extract and represent context from images; thus, the learning process is solely focused on context utilization. A second limitation of systems trained on simulated camera images, such as existing deep learning-based approaches, is a lack of transferability to the real world. Therefore, instead of learning from simulated camera images, this work's learned systems operate on semantic gridmaps which could look identical in the real world or simulation.

Reinforcement Learning

RL, as described in Chapter 2, is a commonly proposed approach for this type of problem [40–44], in which experiences are collected in a simulated environment. However, in this work, the agent's actions do not affect the static environment, and the agent's observations (partial semantic maps) are easy to compute, given a map layout and the robot's position history. This work's approach is a form of model-based learning, but learns from a static dataset instead of environment interaction.

U-Nets for Context Extraction

U-Nets are a DNN architecture originally introduced by Ronneberger et al. for medical image segmentation [45]. More recently, Isola et al. showed that U-Nets can be used for image translation [16, 53]. This work's use of U-Nets is motivated by experiments by Pronobis et al. that show generative networks can imagine unobserved regions of occupancy gridmaps [46]. This suggests that U-Nets can be trained to extract significant geometric context in structured environments. However, unlike [46] where the focus of is on models' abilities to encode context, this work focuses on context utilization for planning.

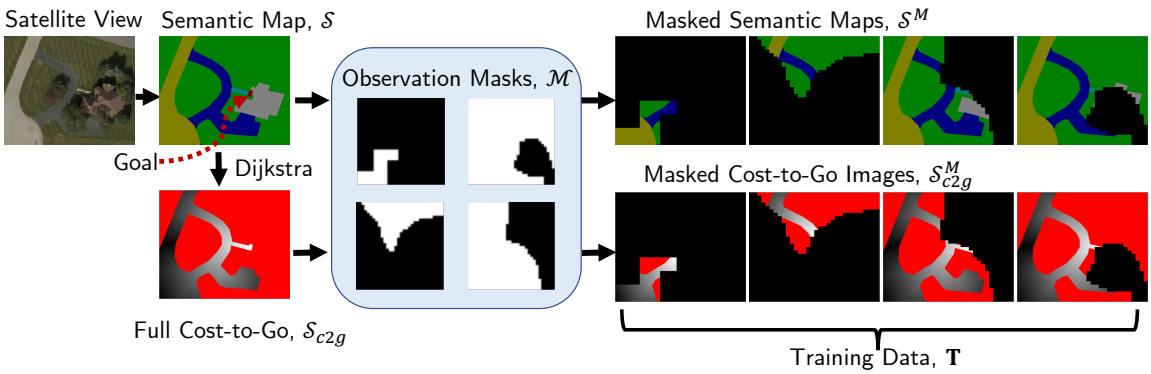


Figure 3-4: Training data creation. A satellite view of a house's front yard is manually converted to a semantic map (top left), with colors for different objects/terrain. Dijkstra's algorithm gives the ground truth distance from the goal to every point along drivable terrain (bottom left) [17]. This cost-to-go is represented in grayscale (lighter near the goal), with red pixels assigned to untraversable regions. To simulate partial map observability, 256 observation masks (center) are applied to the full images to produce the training set.

3.3 Approach

The input to this work's architecture (Fig. 3-2) is a RGB-D camera stream with semantic mask, which is used to produce a partial, top-down, semantic gridmap. An image-to-image translation model is trained to estimate the planning cost-to-go, given the semantic gridmap. Then, the estimated cost-to-go is used to inform a frontier-based exploration planning framework.

3.3.1 Training Data

A typical criticism of learning-based systems is the challenge and cost of acquiring useful training data. Fortunately, domain-specific data already exists for many exploration environments (e.g., satellite images of houses, floor plans for indoor tasks); however, the format differs from data available on a robot with a forward-facing camera. This work uses a mutually compatible representation of gridmaps segmented by terrain class.

A set of satellite images of houses from Bing Maps [54] was manually annotated with terrain labels. The dataset contains 77 houses (31 train, 4 validation, 42 test) from 4 neighborhoods (3 suburban, 1 urban); one neighborhood is purely for training,

Algorithm 1: Automated creation of NN training data

1 **Input:** semantic maps, $\mathcal{S}_{train}$; observability masks, $\mathcal{M}_{train}$
2 **Output:** training image pairs, $\mathcal{T}_{train}$
3 **foreach** $S \in \mathcal{S}_{train}$ **do**
4 $\quad S_{tr} \leftarrow$ Find traversable regions in S
5 $\quad S_{c2g} \leftarrow$ Compute cost-to-go to goal of all pts in S_{tr}
6 $\quad$ **foreach** $M \in \mathcal{M}_{train}$ **do**
7 $\quad\quad S^M_{c2g} \leftarrow$ Apply observation mask M to S_{c2g}
8 $\quad\quad S^M \leftarrow$ Apply observation mask M to S
9 $\quad\quad \mathcal{T}_{train} \leftarrow \{(S^M, S^M_{c2g})\} \cup \mathcal{T}_{train}$

one is split between train/val/test, and the other 2 neighborhoods (including urban) are purely for testing. Each house yields many training pairs, due to the various partial-observability masks applied, so there are 7936 train, 320 validation, and 615 test pairs (explained below). An example semantic map of a suburban front yard is shown in the top left corner of Fig. 3-4.

Algorithm 1 describes the process of automatically generating training data from a set of semantic maps, $\mathcal{S}_{train}$. A semantic map, $S \in S_{train}$, is first separated into traversable (roads, driveways, etc.) and non-traversable (grass, house) regions, represented as a binary array, S_{tr} (Line 4). Then, the shortest path length between each traversable point in S_{tr} and the goal is computed with Dijkstra's algorithm [17] (Line 5). The result, S_{c2g}, is stored in grayscale (Fig. 3-4 bottom left: darker is further from goal). Non-traversable regions are red in S_{c2g}.

This work allows the robot to start with no (or partial) knowledge of a particular environment's semantic map; it observes (uncovers) new areas of the map as it explores the environment. To approximate the partial maps that the robot will have at each planning step, full training maps undergo various masks to occlude certain regions. For some binary observation mask, $M \in \mathcal{M}_{train} \subseteq \{0, 1\}^{h \times w}$: the full map, S, and full cost-to-go, S_{c2g}, are masked with element-wise multiplication as $S^M = S \circ M$ and $S^M_{c2g} = S_{c2g} \circ M$ (Lines 7 and 8). The (input, target output) pairs used to train the image-to-image translator are the set of (S^M, S^M_{c2g}) (Line 9).

3.3.2 Offline Training: Image-to-Image Translation Model

The motivation for using image-to-image translation is that i) a robot's sensor data history can be compressed into an image (semantic gridmap), and ii) an estimate of cost-to-go at every point in the map (thus, an image) enables efficient use of receding-horizon planning algorithms, given only a high-level goal ("front door"). Although the task of learning to predict just the goal location is easier, a cost-to-go estimate implicitly estimates the goal location and then provides substantially more information about how to best reach it.

The image-to-image translator used in this work is based on [16, 53]. The translator is a standard encoder-decoder network with skip connections between corresponding encoder and decoder layers ("U-Net") [45]. The objective is to supply a 256×256 RGB image (semantic map, S) as input to the encoder, and for the final layer of the decoder to output a 256×256 RGB image (estimated cost-to-go map, $\hat{S}_{c2g}$). Three U-Net training approaches are compared: pixel-wise L_1 loss, GAN, and a weighted sum of those two losses, as in [16].

A partial map could be associated with multiple plausible cost-to-gos, depending on the unobserved parts of the map. Without explicitly modeling the distribution of cost-to-gos conditioned on a partial semantic map, $P(\ S^M_{c2g}\ \mid\ S^M)$, the network training objective and distribution of training images are designed so the final decoder layer outputs a most likely cost-to-go-map, $\bar{S}^M_{c2g}$, where

$$\bar{S}^M_{c2g} = \underset{S^M_{c2g} \in \mathbb{R}^{256\times256\times3}}{\operatorname{argmax}} P(S^M_{c2g} \mid S^M). \tag{3.5}$$

3.3.3 Online Mapping: Semantic SLAM

To use the trained network in an online sense-plan-act cycle, the mapping system must produce top-down, semantic maps from the RGB-D camera images and semantic labels available on a robot with a forward-facing depth camera and an image segmentation algorithm [55]. Using [56], the semantic mask image is projected into the world frame using the depth image and camera parameters to produce a point-

Algorithm 2: Deep Cost-to-Go (DC2G) Planner

1 **Input:** current partial semantic map S, pose (p_x, p_y, θ)
2 **Output:** action $\mathbf{u}_t$
3 $S_{tr} \leftarrow$ Find traversable cells in S
4 $S_r \leftarrow$ Find reachable cells in S_{tr} from (p_x, p_y) w/ BFS
5 **if** *goal* $\notin S_r$ **then**
6 $F \leftarrow$ Find frontier cells in S_r
7 $F^e \leftarrow$ Find cells in S_r where $f \in F$ in view
8 $F^e_r \leftarrow S_r \cap F^e$: reachable, frontier-expanding cells
9 $\hat{S}_{c2g} \leftarrow$ Query generator network with input S
10 $C \leftarrow$ Filter and resize $\hat{S}_{c2g}$
11 $(f_x, f_y) \leftarrow \text{argmax}_{f \in F^e_r} C$
12 $\mathbf{u}_{t:\infty} \leftarrow$ Backtrack from (f_x, f_y) to (p_x, p_y) w/ BFS
13 **else**
14 $\mathbf{u}_{t:\infty} \leftarrow$ Shortest path to goal via BFS

cloud, colored by the semantic class of the point in 3-space. Each pointcloud is added to an octree representation, which can be converted to a octomap (3D occupancy grid) on demand, where each voxel is colored by the semantic class of the point in 3-space. Because this work's experiments are 2D, we project the octree down to a 2D semantically-colored gridmap, which is the input to the cost-to-go estimator.

3.3.4 Online Planning: Deep Cost-to-Go

This work's planner is based on the idea of frontier exploration [26], where a frontier is defined as a cell in the map that is observed and traversable, but whose neighbor has not yet been observed. Given a set of frontier cells, the key challenge is in choosing *which* frontier cell to explore next. Existing algorithms often use geometry (e.g., frontier proximity, expected information gain based on frontier size/layout); we instead use context to select frontier cells that are expected to lead toward the destination.

The planning algorithm, called Deep Cost-to-Go (DC2G), is described in Algorithm 2. Given the current partial semantic map, S, the subset of observed cells that are also traversable (road/driveway) is S_{tr} (Line 3). The subset of cells in S_{tr} that are

also reachable, meaning a path exists from the current position, through observed, traversable cells, is S_r (Line 4).

The planner opts to explore if the goal cell is not yet reachable (Line 6). The current partial semantic map, S, is scaled and passed into the image-to-image translator, which produces a 256×256 RGB image of the estimated cost-to-go, $\hat{S}_{c2g}$ (Line 9). The raw output from the U-Net is converted to HSV-space and pixels with high value (not grayscale $\Rightarrow$ estimated not traversable) are filtered out. The remaining grayscale image is resized to match the gridmap's dimensions with a nearest-neighbor interpolation. The value of every grid cell in the map is assigned to be the saturation of that pixel in the translated image (high saturation $\Rightarrow$ "whiter" in grayscale $\Rightarrow$ closer to goal).

To enforce exploration, the only cells considered as possible subgoals are ones which will allow sight beyond frontier cells, F_r^e, (reachable, traversable, and frontier-expanding), based on the known sensing range and FOV. The cell in F_r^e with highest estimated value is selected as the subgoal (Line 11). Since the graph of reachable cells was already searched, the shortest path from the selected frontier cell to the current cell is available by backtracking through the search tree (Line 12). This backtracking procedure produces the list of actions, $\mathbf{u}_{t:\infty}$ that leads to the selected frontier cell. The first action, $\mathbf{u}_t$ is implemented and the agent takes a step, updates its map with new sensor data, and the sense-plan-act cycle repeats. If the goal is deemed reachable, exploration halts and the shortest path to the goal is implemented (Line 14). However, if the goal has been observed, but a traversable path to it does not yet exist in the map, exploration continues in the hope of finding a path to the goal.

A key benefit of the DC2G planning algorithm is that it can be used alongside a local collision avoidance algorithm, which is critical for domains with dynamic obstacles (e.g., pedestrians on sidewalks). This flexibility contrasts end-to-end learning approaches where collision avoidance either must be part of the objective during learning (further increasing training complexity) or must be somehow combined with the policy in a way that differs from the trained policy.

Moreover, a benefit of map creation during exploration is the possibility for the

algorithm to confidently "give up" if it fully explored the environment without finding the goal. This idea would be difficult to implement as part of a non-mapping exploration algorithm, and could address ill-posed exploration problems that exist in real environments (e.g., if no path exists to destination).

3.4 Results

3.4.1 Model Evaluation: Image-to-Image Translation

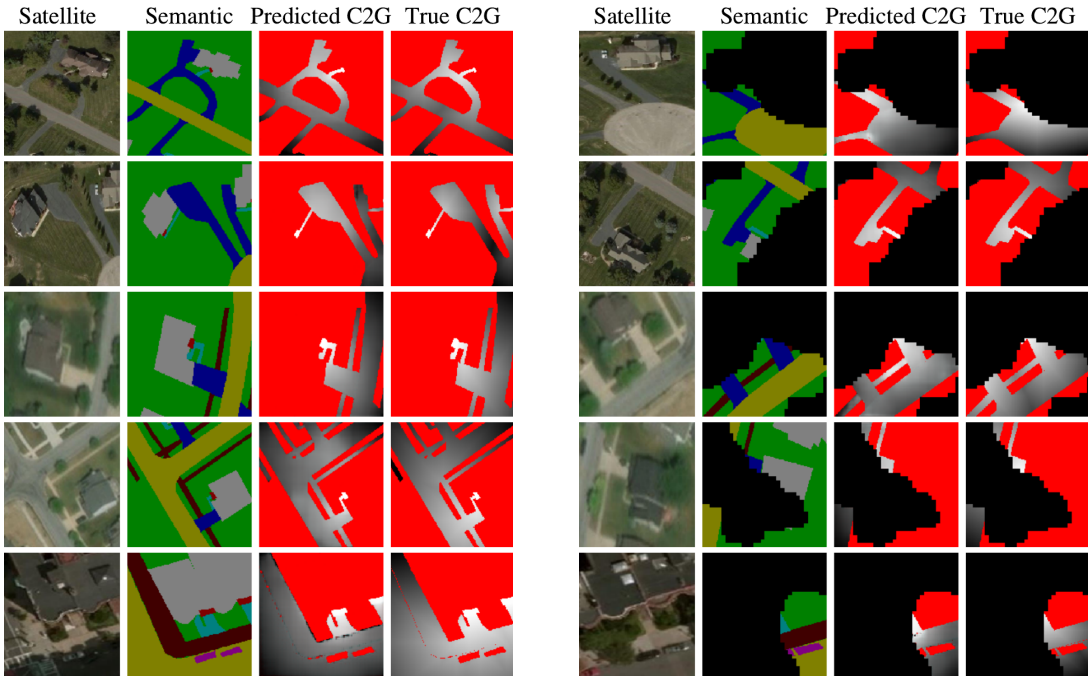
Training the cost-to-go estimator took about 1 hour on a GTX 1060, for 3 epochs (20,000 steps) with batch size of 1. Generated outputs resemble the analytical cost-to-gos within a few minutes of training, but images look sharper/more accurate as training time continues. This notion is quantified in Fig. 3-6, where generated images are compared pixel-to-pixel to the true images in the validation set throughout the training process.

Loss Functions

Networks trained with three different loss functions are compared on three metrics in Fig. 3-6. The network trained with L_1 loss (green) has the best precision/worst recall on identification of regions of low cost-to-go (described below). The GAN achieves almost the same L_1 loss, albeit slightly slower, than the networks trained explicitly to optimize for L_1 loss. Overall, the three networks perform similarly, suggesting the choice of loss function has minimal impact on this work's dataset of low frequency images.

Qualitative Assessment

Fig. 3-5 shows the generator's output on 5 full and 5 partial semantic maps from worlds and observation masks not seen during training. The similarity to the ground truth values qualitatively suggests that the generative network successfully learned the ideas of traversability and contextual clues for goal proximity. Some undesirable



(a) Full Semantic Maps (b) Partial Semantic Maps

Figure 3-5: Qualitative Assessment. Network's predicted cost-to-go on 10 previously unseen semantic maps strongly resembles ground truth. Predictions correctly assign red to untraversable regions, black to unobserved regions, and grayscale with intensity corresponding to distance from goal. Terrain layouts in the top four rows (suburban houses) are more similar to the training set than the last row (urban apartments). Accordingly, network performance is best in the top four rows, but assigns too dark a value in sidewalk/road (brown/yellow) regions of bottom left map, though the traversability is still correct. The predictions in Fig. 3-5b enable planning without knowledge of the goal's location.

features exist, like missed assignment of light regions in the bottom row of Fig. 3-5, which is not surprising because in the training houses (suburban), roads and sidewalks are usually far from the front door, but this urban house has quite different topology.

Quantitative Assessment

In general, quantifying the performance of image-to-image translators is difficult [57]. Common approaches use humans or pass the output into a segmentation algorithm that was trained on real images [57]; but, the first approach is not scalable and the second does not apply here.

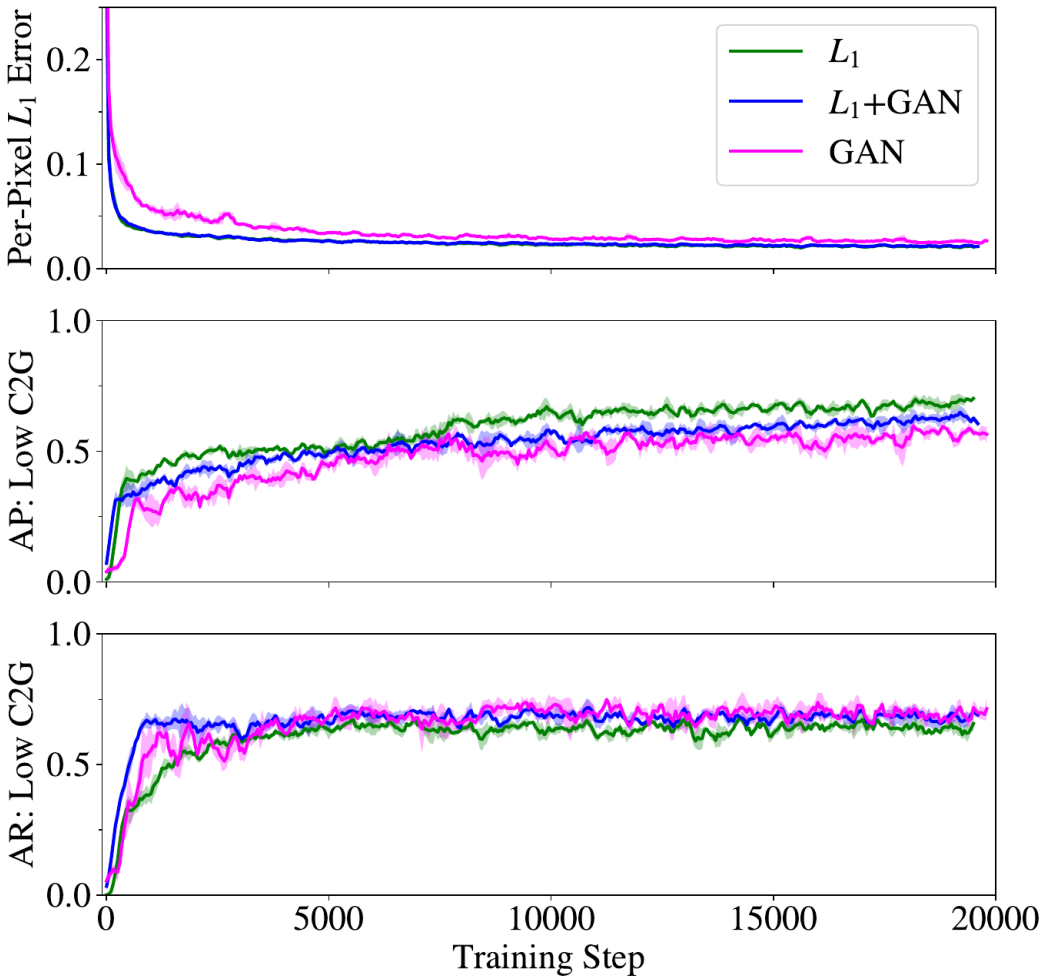


Figure 3-6: Comparing network loss functions. The performance on the validation set throughout training is measured with the standard L_1 loss, and a planning-specific metric of identifying regions of low cost-to-go. The different training loss functions yield similar performance, with slightly better precision/worse recall with L_1 loss (green), and slightly worse L_1 error with pure GAN loss (magenta). 3 training episodes per loss function are individually smoothed by a moving average filter, mean curves shown with $\pm 1\sigma$ shading. These plots show that the choice of loss function has minimal impact on this work's dataset of low frequency images.

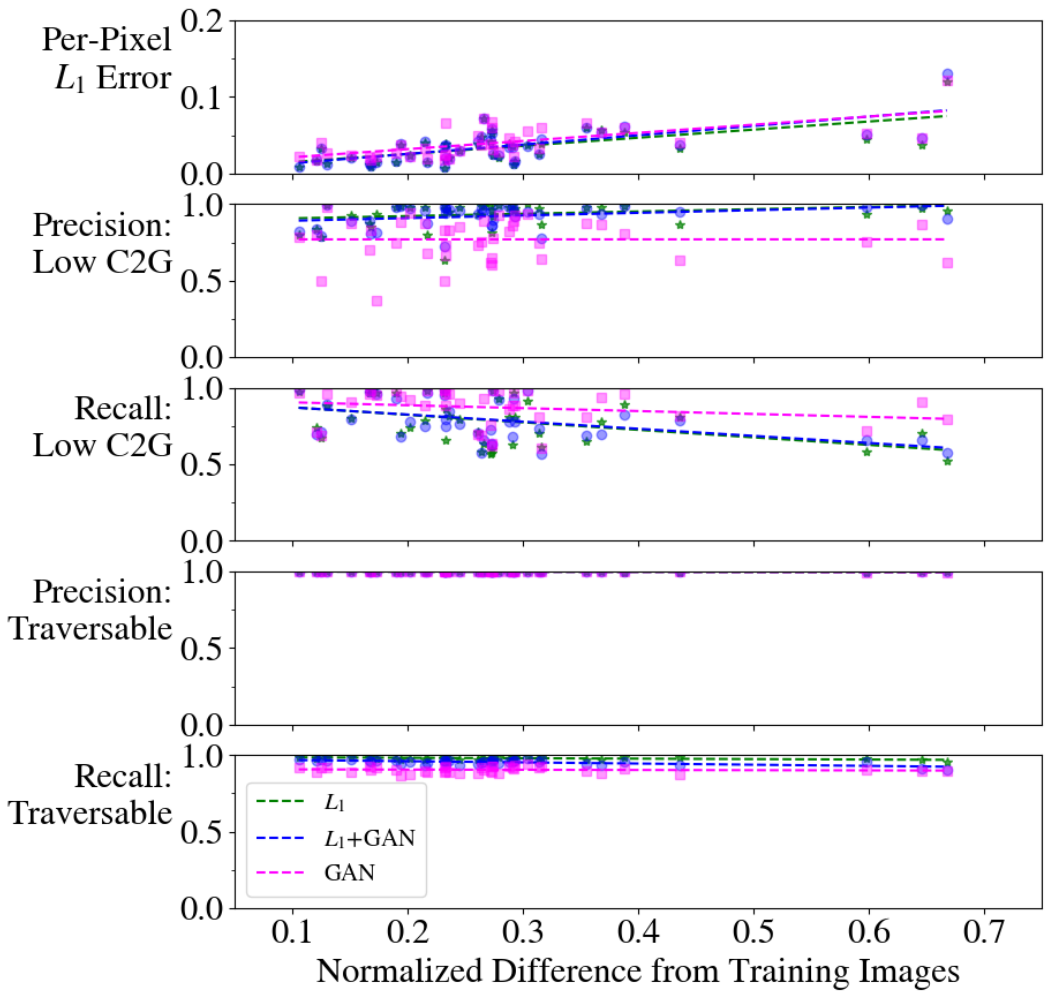


Figure 3-7: Cost-to-Go predictions across test set. Each marker shows the network performance on a full test house image, placed horizontally by the house's similarity to the training houses. Dashed lines show linear best-fit per network. All three networks have similar per-pixel L_1 loss (top row). But, for identifying regions with low cost-to-go (2^{nd}, 3^{rd} rows), GAN (magenta) has worse precision/better recall than networks trained with L_1 loss (blue, green). All networks achieve high performance on estimating traversability (4^{th}, 5^{th} rows). This result could inform application-specific training dataset creation, by ensuring desired test images occur near the left side of the plot.

Unique to this paper's domain[1], for fully-observed maps, ground truth and generated images can be directly compared, since there is a single solution to each pixel's target intensity. Fig. 3-7 quantifies the trained network's performance in 3 ways, on all 42 test images, plotted against the test image's similarity to the training images (explained below). First, the average per-pixel L_1 error between the predicted and true cost-to-go images is below 0.15 for all images. To evaluate on a metric more related to planning, the predictions and targets are split into two categories: pixels that are deemed traversable (HSV-space: $S < 0.3$) or not. This binary classification is just a means of quantifying how well the network learned a relevant sub-skill of cost-to-go prediction; the network was not trained on this objective. Still, the results show precision above 0.95 and recall above 0.8 for all test images. A third metric assigns pixels to the class "low cost-to-go" if sufficiently bright (HSV-space: $V > 0.9 \wedge S < 0.3$). Precision and recall on this binary classification task indicate how well the network finds regions that are close to the destination. The networks perform well on precision (above 0.75 on average), but not as well on recall, meaning the networks missed some areas of low cost-to-go, but did not spuriously assign many low cost-to-go pixels. This is particularly evident in the bottom row of Fig. 3-5a, where the path leading to the door is correctly assigned light pixels, but the sidewalk and road are incorrectly assigned darker values.

To measure generalizability in cost-to-go estimates, the test images are each assigned a similarity score $\in [0, 1]$ to the closest training image. The score is based on Bag of Words with custom features[2], computed by breaking the training maps into grids, computing a color histogram per grid cell, then clustering to produce a vocabulary of the top 20 features. Each image is then represented as a normalized histogram of words, and the minimum L_1 distance to a training image is assigned that test image's score. This metric captures whether a test image has many colors/small regions in common with a training image.

[1]As opposed to popular image translation tasks, like sketch-to-image, or day-to-night, where there is a distribution of acceptable outputs.

[2]The commonly-used SIFT, SURF, and ORB algorithms did not provide many features in this work's low-texture semantic maps

The network performance on L_1 error, and recall of low cost-to-go regions, declines as images differ from the training set, as expected. However, traversabilty and precision of low cost-to-go regions were not sensitive to this parameter.

Without access to the true distribution of cost-to-go maps conditioned on a partial semantic map, this work evaluates the network's performance on partial maps indirectly, by measuring the *planner's* performance, which would suffer if given poor cost-to-go estimates.

3.4.2 Low-Fidelity Planner Evaluation: Gridworld Simulation

The low-fidelity simulator uses a 50×50-cell gridworld [58] to approximate a real robotic vehicle operating in a delivery context. Each grid cell is assigned a static class (house, driveway, etc.); this terrain information is useful both as context to find the destination, but also to enforce the real-world constraint that robots should not drive across houses' front lawns. The agent can read the type of any grid cell within its sensor FOV (to approximate a common RGB-D sensor: 90° horizontal, 8-cell radial range). To approximate a SLAM system, the agent remembers all cells it has seen since the beginning of the episode. At each step, the agent sends an observation to the planner containing an image of the agent's semantic map knowledge, and the agent's position and heading. The planner selects one of three actions: go forward, or turn ±90°.

Each gridworld is created by loading a semantic map of a real house from Bing Maps, and houses are categorized by the 4 test neighborhoods. A random house and starting point on the road are selected 100 times for each of the 4 neighborhoods. The three planning algorithms compared are DC2G, Frontier [26] which always plans to the nearest frontier cell (pure exploration), and an oracle (with full knowledge of the map ahead of time).

DC2G and Frontier performance is quantified in Fig. 3-8 by inverse path length. For each test episode i, the oracle reaches the goal in l_i^* steps, and the agent following a different planner reaches the goal in $l_i \leq l_i^*$ steps. The inverse path length is thus $\frac{l_i^*}{l_i}$, and the distribution of inverse path lengths is shown as a boxplot for the 100 scenarios

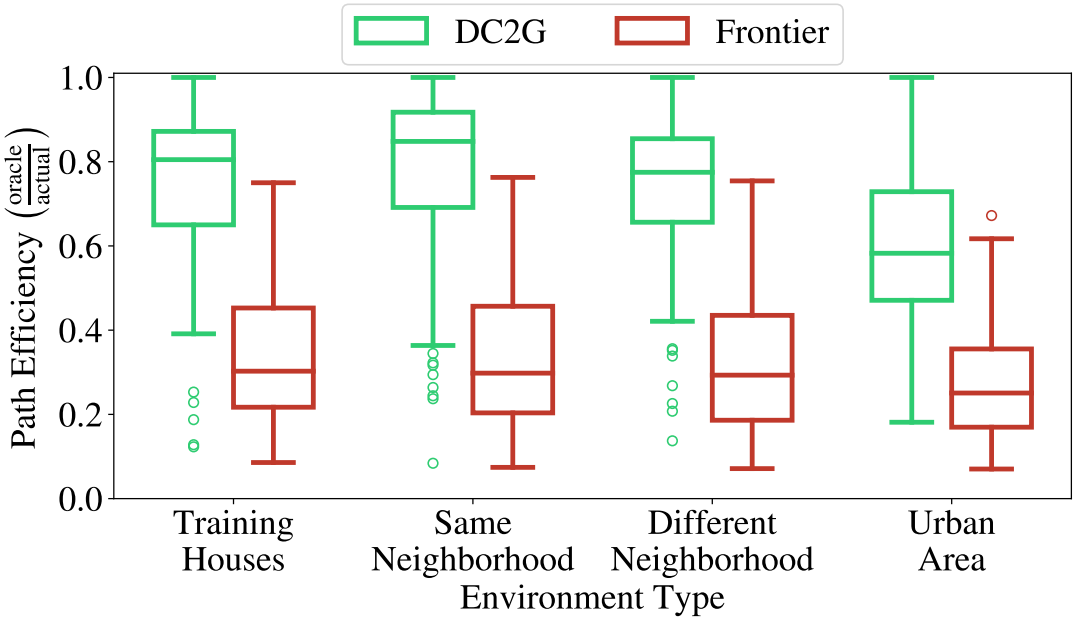


Figure 3-8: Planner performance across neighborhoods. DC2G reaches the goal faster than Frontier [26] by prioritizing frontier points with a learned cost-to-go prediction. Performance measured by "path efficiency", or Inverse Path Length (IPL) per episode, $IPL = \frac{\text{optimal path length}}{\text{planner path length}}$. The simulation environments come from real house layouts from Bing Maps, grouped by the four neighborhoods in the dataset, showing DC2G plans generalize beyond houses seen in training.

per test neighborhood. Because the DC2G and Frontier planners are always expanding the map (planning to a frontier point), we see 100% success rate in these scenarios, which makes average inverse path length equivalent to Success Weighted by Inverse Path Length (SPL), a performance measure for embodied navigation recommended by [47].

Fig. 3-8 is grouped by neighborhood to demonstrate that DC2G improved the plans across many house types, beyond the ones it was trained on. In the left category, the agent has seen these houses in training, but with different observation masks applied to the full map. In the right three categories, both the houses and masks are new to the network. Although grouping by neighborhood is not quantitative, the test-to-train image distance metric (above) was not correlated with planner performance, suggesting other factors affect plan lengths, such as topological differences in layout that were not quantified in this work.

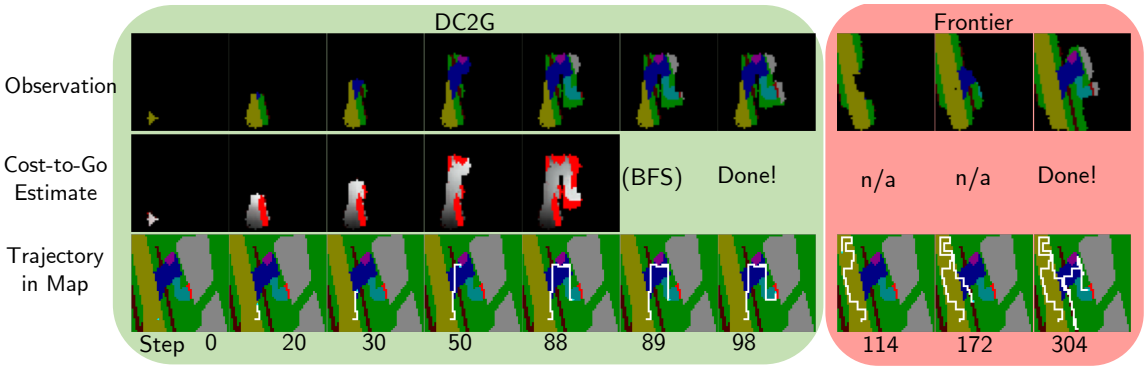


Figure 3-9: Sample gridworld scenario. The top row is the agent's observed semantic map (network input); the middle is its current estimate of the cost-to-go (network output); the bottom is the trajectory so far. The DC2G agent reaches the goal much faster (98 vs. 304 steps) by using learned context. At DC2G (green panel) step 20, the driveway appears in the semantic map, and the estimated cost-to-go correctly directs the search in that direction, then later up the walkway (light blue) to the door (red). Frontier (pink panel) is unaware of typical house layouts, and unnecessarily explores the road and sidewalk regions thoroughly before eventually reaching the goal.

Across the test set of 42 houses, DC2G reaches the goal within 63% of optimal, and 189% faster than Frontier on average[3].

3.4.3 Planner Scenario

A particular trial is shown in Fig. 3-9 to give insight into the performance improvement from DC2G. Both algorithms start in the same position in a world that was not seen during training. The top row shows the partial semantic map at that timestep (observation), the middle row is the generator's cost-to-go estimate, and the bottom shows the agent's trajectory in the whole, unobservable map. Both algorithms begin with little context since most of the map is unobserved. At step 20, DC2G (green box, left) has found the intersection between road (yellow) and driveway (blue): this context causes it to turn up the driveway. By step 88, DC2G has observed the goal cell, so it simply plans the shortest path to it with BFS, finishing in 98 steps. Conversely, Frontier (pink box, right) takes much longer (304 steps) to reach the

[3]Note that as the world size increases, these path length statistics would appear to make the Frontier planner seem even worse relative to optimal, since the optimal path would likely grow in length more slowly than the number of cells that must be explored.

goal, because it does not consider terrain context, and wastes many steps exploring road/sidewalk areas that are unlikely to contain a house's front door.

3.4.4 Unreal Simulation & Mapping

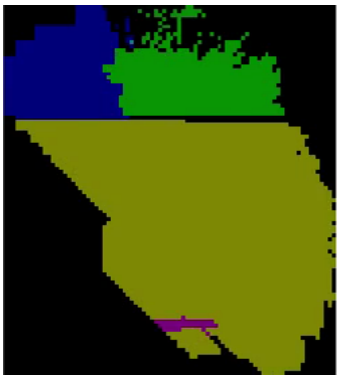
The gridworld is sufficiently complex to demonstrate the fundamental limitations of a baseline algorithm, and also allows quantifiable analysis over many test houses. However, to navigate in a real delivery environment, a robot with a forward-facing camera needs additional capabilities not apparent in a gridworld. Therefore, this work demonstrates the algorithm in a high-fidelity simulation of a neighborhood, using AirSim [59] on Unreal Engine. The simulator returns camera images in RGB, depth, and semantic mask formats, and the mapping software described in Section 3.3 generates the top-down semantic map.

A rendering of one house in the neighborhood is shown in Fig. 3-10a. This view highlights the difficulty of the problem, since the goal (front door) is occluded by the trees, and the map in Fig. 3-10b is rather sparse. The semantic maps from the mapping software are much noisier than the perfect maps the network was trained with. Still, the predicted cost-to-go in Fig. 3-10c is lightest in the driveway area, causing the planned path in Fig. 3-10d (red) from the agent's starting position (cyan) to move up the driveway, instead of exploring more of the road. The video shows trajectories of two delivery simulations: `https://youtu.be/yVlnbqEFct0`.

3.4.5 Comparison to RL

We now discuss how an RL formulation compares to our exploration-based planning with a learned context approach. We trained a DQN [60] agent in one of the houses in the dataset (`worldn000m001h001`) for 4400 episodes. At each step, the RL agent has access to the partial semantic gridmap (2D image) and scalars corresponding to the agent's position in the gridmap, and its heading angle. We did not succeed in training the agent with just these components as network inputs, but instead pre-processed the observation to assign a white color the agent's current cell in the gridmap. Thus,

(a) Unreal Engine Rendering of a Last-Mile Delivery



(b) Semantic Map

(c) Predicted Cost-to-Go

(d) Planned Path

Figure 3-10: Unreal Simulation. Using a network trained only on aerial images, the planner guides the robot toward the front door of a house not seen before, with a forward-facing camera. The planned path (d) starts from the robot's current position (cyan) along grid cells (red) to the frontier-expanding point (green) with lowest estimated cost-to-go (c), up the driveway.

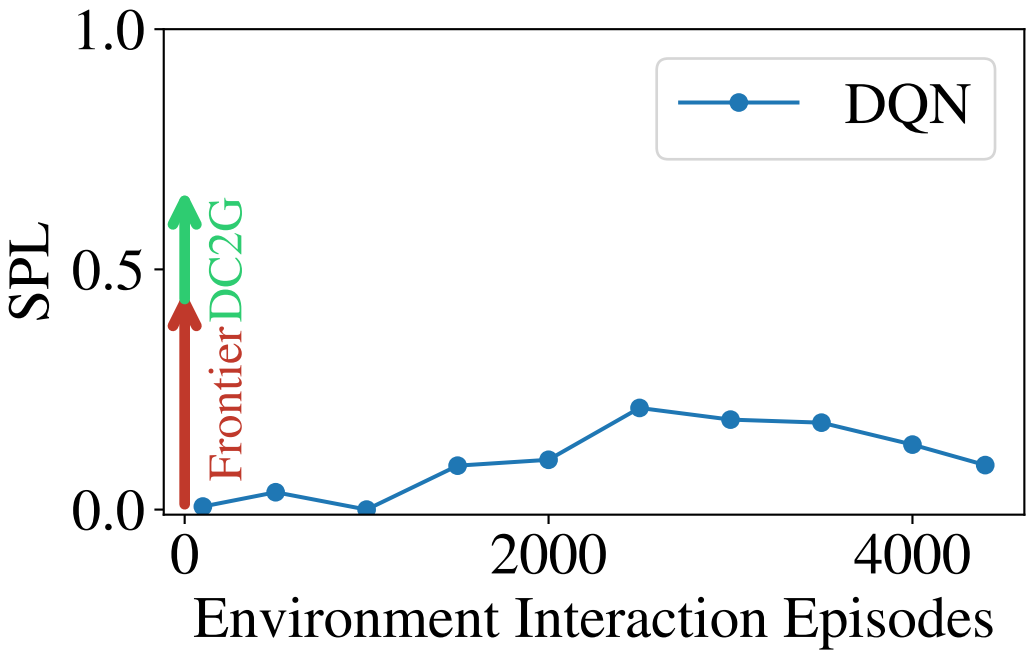


Figure 3-11: DC2G vs. RL. In blue, an RL agent is trained for 4400 episodes to increase its SPL (efficiency in reaching the goal). With zero training, the Frontier [26] algorithm provides better SPL (red). Instead of training in the environment, DC2G trains on an offline dataset to further improve the performance of a Frontier-based planner. This suggests that planning-based methods could be more effective than RL in these types of navigation problems.

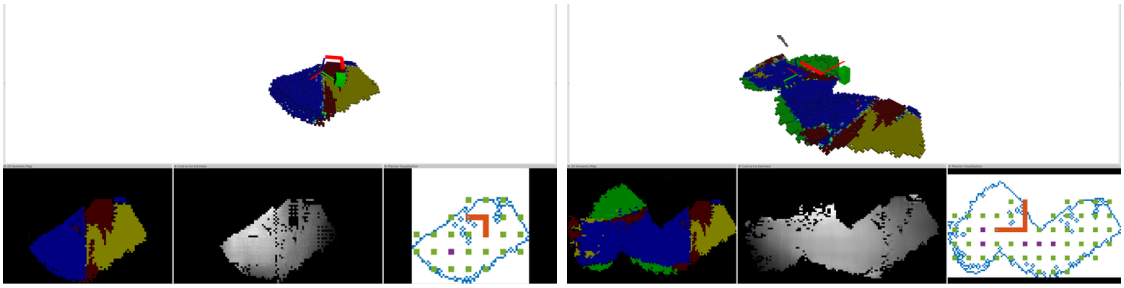
the observation consists of the processed partial semantic map and the agent's current heading. The RL agent chooses one of the 3 discrete actions described earlier.

The reward function is defined as: if the current cell is traversable, the cost-to-go from the current position according to Dijkstra's algorithm [17]; if the current cell is non-traversable, -1. We did not succeed in training the agent with a sparse reward function.

We quantify the performance of the RL agent throughout training in Fig. 3-11. The blue curve shows that SPL in 100 random initializations at one house increases as the number of training episodes increases, but begins to drop off after about 2500 episodes. RL achieves a maximum SPL of about 0.25. On the other hand, simply using the Frontier algorithm *with zero training* leads to about 0.45 SPL. Moreover, our DC2G adds context-specific knowledge on top of the Frontier planner, providing further improvement in SPL, *with zero interactions in the environment.*

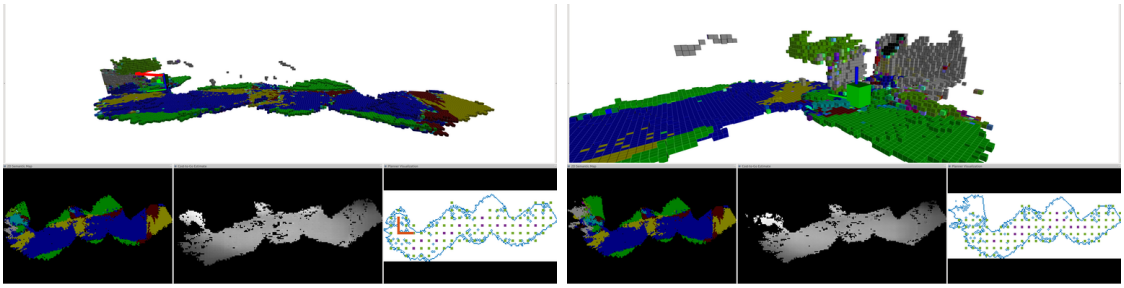
Furthermore, we did not see an ability to generalize the RL knowledge to other houses, nor could we successfully train the RL agent on more than 2 houses simultaneously. While state-of-art multi-task RL algorithms might be better suited to handle the generalizability problem than DQN, this comparison raises more fundamental questions of whether RL is an appropriate framework for this type of navigation problem.

The reward function we used (and presumably other reward functions) is computed using knowledge of the entire world map and goal position. This is the same information used in the supervised learning formulation we propose, but supervised learning can transmit the information encoded in the reward function much more efficiently to the policy than RL can. The key difference is that the RL agent only obtains information about the states visited on a particular trajectory, whereas the supervised learning agent obtains information about all states in the partial semantic map at each learning step.



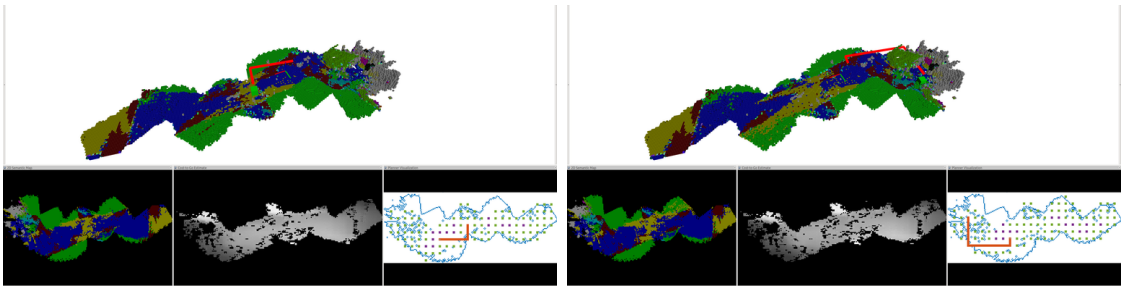
(a) Beginning of Trajectory

(b) Explore toward gap in driveway



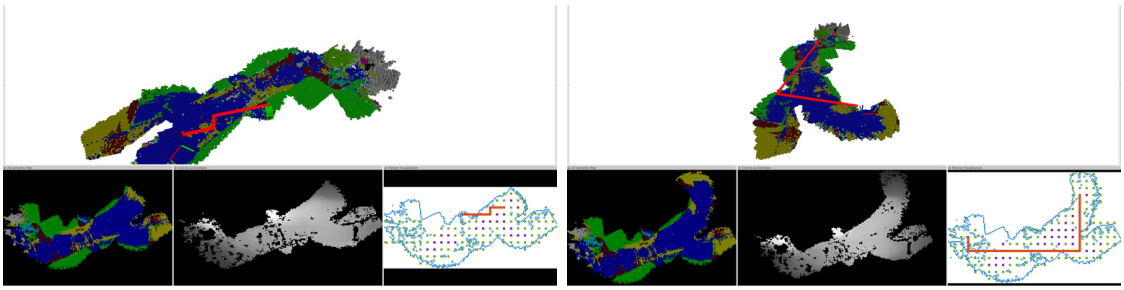
(c) 👍 On walkway, goal at end of walkway

(d) 👍 At front door, goal nearby



(e) 👎 Leaving house, goal on sidewalk

(f) 👍 Leaving house, goal on walkway



(g) 👎 Back in street, goal on sidewalk

(h) 👍 Back in street, goal on walkway

Figure 3-12: DC2G on Real Robot Data at Real House. For 8 timesteps, the 3D semantic octomap is shown with a coordinate axis corresponding to the robot's current position, DC2G's recommended frontier point shown as a green box, and a red line showing a path to the recommended frontier point. The bottom rows show the top-down 2D semantic map, the estimated cost-to-go, and the planner's path (blue is map frontier, green is frontier-expanding points, orange is path to DC2G goal). At some timesteps (c, d, f, h), DC2G correctly recommends to drive toward the house's front door. Fixing the inaccuracies in the semantic map (e.g., some classes are labeled incorrectly) might improve some of the failure cases (e, g).

3.4.6 Data from Hardware Platform

To show how the proposed algorithm could be eventually applied to a hardware platform, Fig. 3-12 shows example motion plans based on data collected from a Jackal robot [61] and wheel encoders for odometry and a Realsense D435 camera for perception. The dataset's robot was teleoperated at a real house in a suburban neighborhood in Michigan. Thus, the analysis provided here shows where the robot *would have* driven had we closed the loop and replaced the teleoperator with the DC2G algorithm.

To produce the semantic maps using real sensor data, we needed a semantic segmentation system. We trained [62] on a custom dataset of urban house and neighborhood images from forward-facing cameras at about car level. This dataset was annotated with driveway, road, sidewalk, grass, house, tree, car, walkway classes, among other minor ones.

The trained semantic segmentation system provides correct semantic masks some of the time, but certain classes are very difficult to distinguish (both for human annotators and the trained network), such as driveway vs. sidewalk vs. walkway. We did not quantify this performance, but the semantic maps in Fig. 3-12 provide some indication of the inaccuracies, for instance the driveway is not completely blue (some yellow, maroon).

Because DC2G was trained on satellite images with high-quality annotations, the inaccuracies in the semantic maps cause the cost-to-go estimates to be inaccurate some of the time as well.

3.4.7 Discussion

It is important to note that DC2G is expected to perform worse than Frontier if the test environment differs significantly from the training set, since context is task-specific. The dataset covers several neighborhoods, but in practice, the training dataset could be expanded/tailored to application-specific preferences. In this dataset, Frontier outperformed DC2G on 2/77 houses. However, even in case of

Frontier outperforming DC2G, DC2G is still guaranteed to find the goal eventually (after exploring all frontier cells), unlike a system trained end-to-end that could get stuck in a local optimum.

The DC2G algorithm as described requires knowledge of the environment's dimensions; however, in practice, a maximum search area would likely already be defined relative to the robot's starting position (e.g., to ensure operation time less than battery life).

3.5 Summary

This work presented an algorithm for learning to utilize context in a structured environment in order to inform an exploration-based planner. The new approach, called Deep Cost-to-Go (DC2G), represents scene context in a semantic gridmap, learns to estimate which areas are beneficial to explore to quickly reach the goal, and then plans toward promising regions in the map. The efficient training algorithm requires zero training in simulation: a context extraction model is trained on a static dataset, and the creation of the dataset is highly automated. The algorithm outperforms pure frontier exploration by 189% across 42 real test house layouts. A high-fidelity simulation shows that the algorithm can be applied on a robot with a forward-facing camera to successfully complete last-mile deliveries.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Collision Avoidance in Pedestrian-Rich Environments with Deep Reinforcement Learning

4.1 Introduction

A fundamental challenge in autonomous vehicle operation is to safely negotiate interactions with other dynamic agents in the environment. For example, it is important for self-driving cars to take other vehicles' motion into account, and for delivery robots to avoid colliding with pedestrians. While there has been impressive progress in the past decade [63], fully autonomous navigation remains challenging, particularly in uncertain, dynamic environments cohabited by other mobile agents. The challenges arise because the other agents' intents and policies (i.e., goals and desired paths) are typically not known to the planning system, and, furthermore, explicit communication of such hidden quantities is often impractical due to physical limitations. These issues motivate the use of decentralized collision avoidance algorithms.

Existing work on decentralized collision avoidance can be classified into cooperative and non-cooperative methods. Non-cooperative methods first predict the other agents' motion and then plan a collision-free path for the vehicle with respect to the

other agents' predicted motion. However, this can lead to the *freezing robot problem* as coined by Trautman and Krause [64], where the vehicle fails to find any feasible path because the other agents' predicted paths would occupy a large portion of the traversable space. Cooperative methods address this issue by modeling interaction in the planner, such that the vehicle's action can influence the other agent's motion, thereby having all agents share the responsibility for avoiding collision. Cooperative methods include reaction-based methods [65–68] and trajectory-based methods [69–71].

This work seeks to combine the best of both types of cooperative techniques – the computational efficiency of reaction-based methods and the smooth motion of trajectory-based methods. To this end, the work presents the Collision Avoidance with Deep Reinforcement Learning (CADRL) algorithm, which tackles the aforementioned trade-off between computation time and smooth motion by using Reinforcement Learning (RL) to offload the expensive online computation to an offline learning procedure. Specifically, a computationally efficient (i.e., real-time implementable) interaction rule is developed by learning a policy that implicitly encodes cooperative behaviors.

Learning the collision avoidance policy for CADRL presents several challenges. A first key challenge is that the number of other agents in the environment can vary between timesteps or experiments, however the typical feedforward neural networks used in this domain require a fixed-dimension input. Our prior work defines a maximum number of agents that the network can observe, and other approaches use raw sensor data as the input [72, 73]. This work instead uses an idea from Natural Language Processing [74, 75] to encode the varying size state of the world (e.g., positions of other agents) into a fixed-length vector, using Long Short-Term Memory (LSTM) [76] cells at the network input. This enables the algorithm to make decisions based on an arbitrary number of other agents in the robot's vicinity.

A second fundamental challenge is in finding a policy that makes realistic assumptions about other agents' belief states, policies, and intents. This work learns a collision avoidance policy without assuming that the other agents follow any par-

ticular behavior model and without explicit assumptions on homogeneity [72] (e.g., agents of the same size and nominal speed) or specific motion models (e.g., constant velocity) over short timescales [19, 20].

The main contributions of this work are (i) a new collision avoidance algorithm that greatly outperforms prior works as the number of agents in the environments is increased: a key factor in that improvement is to relax the assumptions on the other agents' behavior models during training and inference, (ii) a LSTM-based strategy to address the challenge that the number of neighboring agents could be large and could vary in time, (iii) simulation results that show significant improvement in solution quality compared with previous state-of-the-art methods, and (iv) hardware experiments with aerial and ground robots to demonstrate that the proposed algorithm can be deployed in real time on robots with real sensors. Open-source software based on this manuscript includes a pre-trained collision avoidance policy (as a ROS package), `cadrl_ros`[1], and a simulation/training environment with several implemented policies, `gym_collision_avoidance`[2]. Videos of the experimental results are posted at `https://youtu.be/Bjx4ZEov0yE`.

4.2 Background

4.2.1 Problem Formulation

The non-communicating, multiagent collision avoidance problem can be formulated as a sequential decision making problem [19, 20]. In an n-agent scenario ($\mathbb{N}_{\leq n} = \{1, 2, \ldots, n\}$), denote the joint world state, $\mathbf{s}_t^{jn}$, agent i's state, $\mathbf{s}_{i,t}$, and agent i's action, $\mathbf{u}_{i,t}$, $\forall i \in \mathbb{N}_{\leq n}$. Each agent's state vector is composed of an observable and unobservable (hidden) portion, $\mathbf{s}_{i,t} = [\mathbf{s}_{i,t}^o, \mathbf{s}_{i,t}^h]$. In the global frame, observable states are the agent's position, velocity, and radius, $\mathbf{s}^o = [p_x, p_y, v_x, v_y, r] \in \mathbb{R}^5$, and unobservable states are the goal position, preferred speed, and orientation[3], $\mathbf{s}^h =$

[1] `https://github.com/mit-acl/cadrl_ros`

[2] `https://github.com/mit-acl/gym-collision-avoidance`

[3] Other agents' positions and velocities are straightforward to estimate with a 2D Lidar, unlike human body heading angle

$[p_{gx}, p_{gy}, v_{pref}, \psi] \in \mathbb{R}^4$. The action is a speed and heading angle, $\mathbf{u}_t = [v_t, \psi_t] \in \mathbb{R}^2$. The observable states of all $n-1$ other agents is denoted, $\tilde{\mathbf{S}}^o_{i,t} = \{\tilde{\mathbf{s}}^o_{j,t} : j \in \mathbb{N}_{\leq n} \setminus i\}$. A policy, $\pi : \left(\mathbf{s}_{0:t}, \tilde{\mathbf{S}}^o_{0:t}\right) \mapsto \mathbf{u}_t$, is developed with the objective of minimizing expected time to goal $\mathbb{E}[t_g]$ while avoiding collision with other agents,

$$\underset{\pi_i}{\text{argmin}} \quad \mathbb{E}\left[t_g | \mathbf{s}_i, \tilde{\mathbf{S}}^o_i, \pi_i\right] \tag{4.1}$$

$$s.t. \quad ||\mathbf{p}_{i,t} - \tilde{\mathbf{p}}_{j,t}||_2 \geq r_i + r_j \qquad \forall j \neq i, \forall t \tag{4.2}$$

$$\mathbf{p}_{i,t_g} = \mathbf{p}_{i,g} \quad \forall i \tag{4.3}$$

$$\mathbf{p}_{i,t} = \mathbf{p}_{i,t-1} + \Delta t \cdot \pi_i(\mathbf{s}_{i,t-1}, \tilde{\mathbf{S}}^o_{i,t-1}) \forall i, \tag{4.4}$$

where Eq. (4.2) is the collision avoidance constraint, Eq. (4.3) is the goal constraint, Eq. (4.4) is the agents' kinematics, and the expectation in Eq. (4.1) is with respect to the other agent's unobservable states (intents) and policies.

Although it is difficult to solve for the optimal solution of Eq. (4.1)-Eq. (4.4), this problem formulation can be useful for understanding the limitations of the existing methods. In particular, it provides insights into the approximations/assumptions made by existing works.

4.2.2 Related Work

Most approaches to collision avoidance with dynamic obstacles employ Model Predictive Control (MPC) [77] in which a planner selects a minimum cost action sequence, $\mathbf{u}_{i,t:t+T}$, using a prediction of the future world state, $P(\mathbf{s}^{jn}_{t+1:t+T+1} | \mathbf{s}^{jn}_{0:t}, \mathbf{u}_{i,t:t+T})$, conditioned on the world state history, $\mathbf{s}^{jn}_{0:t}$. While the first actions in the sequence are being implemented, the subsequent action sequence is updated by re-planning with the updated world state information (e.g., from new sensor measurements). The prediction of future world states is either prescribed using domain knowledge (model-based approaches) or learned from examples/experiences (learning-based approaches).

Model-based approaches

Early approaches model the world as a static entity, $[v_x, v_y] = \mathbf{0}$, but replan quickly to try to capture the motion through updated (p_x, p_y) measurements, such as the Dynamic Window Approach by Fox et al. [78]. This leads to time-inefficient paths among dynamic obstacles, since the planner's world model does not anticipate future changes in the environment due to the obstacles' motion.

To improve the predictive model, *reaction-based* methods use one-step interaction rules based on geometry or physics to ensure collision avoidance. These methods (Van den Berg et al. [67], Ferrer et al. [66], and Alonso-Mora et al. [68]) often specify a Markovian policy, $\pi(\mathbf{s}_{0:t}^{jn}) = \pi(\mathbf{s}_t^{jn})$, that optimizes a one-step cost while satisfying collision avoidance constraints. For instance, in velocity obstacle approaches [67, 68], an agent chooses a collision-free velocity that is closest to its preferred velocity (i.e., directed toward its goal). Given this one-step nature, reaction-based methods do account for current obstacle motion, but do not anticipate the other agents' hidden intents – they instead rely on a fast update rate to react quickly to the other agents' changes in motion. Although computationally efficient given these simplifications, reaction-based methods are myopic in time, which can sometimes lead to generating unnatural trajectories [19, 70].

Trajectory-based methods compute plans on a longer timescale to produce smoother paths but are often computationally expensive or require knowledge of unobservable states. A subclass of non-cooperative approaches (Phillips et al. [80] and Aoude et al. [81]) propagates the other agents' dynamics forward in time and then plans a collision-free path with respect to the other agents' predicted paths. However, in crowded environments, the set of predicted paths could occupy a large portion of the space, which leads to the *freezing robot problem* [64]. A key to resolving this issue is to account for interactions, such that each agent's motion can affect one another. Thereby, a subclass of cooperative approaches (Kretzschmar et al. [69], Trautman et al. [70], and Kuderer et al. [71]) has been proposed, which solve Eq. (4.1)-Eq. (4.4) in two steps. First, the other agents' hidden states (i.e., goals) are inferred from

their observed trajectories, $\hat{\tilde{\mathbf{S}}}_t^h = f(\tilde{\mathbf{S}}_{0:t}^o)$, where $f(\cdot)$ is a inference function. Second, a centralized path planning algorithm, $\pi(\mathbf{s}_{0:t}, \tilde{\mathbf{S}}_{0:t}^o) = \pi_{central}(\mathbf{s}_t, \tilde{\mathbf{S}}_t^o, \hat{\tilde{\mathbf{S}}}_t^h)$, is employed to find jointly feasible paths. By planning/anticipating complete paths, trajectory-based methods are no longer myopic. However, both the inference and the planning steps are computationally expensive, and need to be carried out online at each new observation (sensor update $\tilde{\mathbf{S}}_t^o$). Another recent trajectory-based approach reasons about multiagent path topologies [82].

Learning-based approaches

Our recent works [19, 20] proposed a third category that uses a reinforcement learning framework to solve Eq. (4.1)-Eq. (4.4). As in the reactive-based methods, we make a Markovian assumption: $\pi(\mathbf{s}_{0:t}^{jn}) = \pi(\mathbf{s}_t^{jn})$. The expensive operation of modeling the complex interactions is learned in an offline training step, whereas the learned policy can be queried quickly online, combining the benefits of both reactive- and trajectory-based methods. Our prior methods pre-compute a value function, $V(\mathbf{s}^{jn})$, that estimates the expected time to the goal from a given configuration, which can be used to select actions using a one-step lookahead procedure described in those works. To avoid the lookahead procedure, this work directly optimizes a policy $\pi(\mathbf{s}^{jn})$ to select actions to minimize the expected time to the goal. The differences from other learning-based approaches will become more clear after a brief overview of reinforcement learning in this problem.

4.2.3 Reinforcement Learning

The following provides a RL formulation (see Chapter 2) of the n-agent collision avoidance problem relating to Eq. (4.1)-Eq. (4.4).

State space The joint world state, $\mathbf{s}^{jn}$, was defined in Section 4.2.1.

Action space The choice of action space depends on the vehicle model. A natural choice of action space for differential drive robots is a linear and angular speed (which

can be converted into wheel speeds), that is, $\mathbf{u} = [s, \omega]$. The action space is either discretized directly, or represented continuously by a function of discrete parameters.

Reward function A sparse reward function is specified to award the agent for reaching its goal Eq. (4.3), and penalize the agent for getting too close or colliding with other agents Eq. (4.2),

$$R(\mathbf{s}^{jn}, \mathbf{u}) = \begin{cases} 1 & \text{if } \mathbf{p} = \mathbf{p}_g \\ -0.1 + d_{min}/2 & \text{if } 0 < d_{min} < 0.2 \\ -0.25 & \text{if } d_{min} < 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where d_{min} is the distance to the closest other agent. Optimizing the hyperparameters (e.g., -0.25) in R_{col} is left for future work. Note that we use discount $\gamma < 1$ to encourage efficiency instead of a step penalty.

State transition model A probabilistic state transition model, $P(\mathbf{s}_{t+1}^{jn} | \mathbf{s}_t^{jn}, \mathbf{u}_t)$, is determined by the agents' kinematics as defined in Eq. (4.4). Since the other agents' actions also depend on their policies and hidden intents (e.g., goals), the system's state transition model is unknown.

Value function One method to find the optimal policy is to first find the optimal value function,

$$V^*(\mathbf{s}_0^{jn}) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \, R(\mathbf{s}_t^{jn}, \pi^*(\mathbf{s}_t^{jn}))\right], \tag{4.6}$$

where $\gamma \in [0, 1)$ is a discount factor. Many methods exist to estimate the value function in an offline training process [24].

Deep Reinforcement Learning To estimate the high-dimensional, continuous value function (and/or associated policy), it is common to approximate with a Deep

Neural Network (DNN) parameterized by weights and biases, θ, as in [60]. This work's notation drops the parameters except when required for clarity, e.g., $V(\mathbf{s};\theta) = V(\mathbf{s})$.

Decision-making Policy A value function of the current state can be implemented as a policy,

$$\pi^*(\mathbf{s}_{t+1}^{jn}) = \underset{\mathbf{u}}{\operatorname{argmax}}\, R(\mathbf{s}_t, \mathbf{u}) + \gamma^{\Delta t \cdot v_{pref}} \int_{\mathbf{s}_{t+1}^{jn}} P(\mathbf{s}_t^{jn}, \mathbf{s}_{t+1}^{jn} | \mathbf{u}) V^*(\mathbf{s}_{t+1}^{jn}) d\mathbf{s}_{t+1}^{jn}. \tag{4.7}$$

Our previous works avoid the complexity in explicitly modeling $P(\mathbf{s}_{t+1}^{jn}|\mathbf{s}_t^{jn}, \mathbf{u})$ by assuming that other agents continue their current velocities, $\hat{\mathbf{V}}_t$, for a duration Δt, meaning the policy can be extracted from the value function,

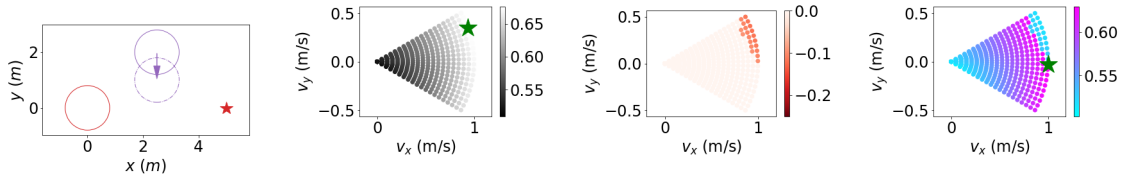
$$\hat{\mathbf{s}}_{t+1,\mathbf{u}}^{jn} \leftarrow [f(\mathbf{s}_t, \Delta t \cdot \mathbf{u}), f(\tilde{\mathbf{S}}_t^o, \Delta t \cdot \hat{\mathbf{V}}_t)] \tag{4.8}$$

$$\pi_{CADRL}(\mathbf{s}_t^{jn}) = \underset{\mathbf{u}}{\operatorname{argmax}}\, R_{col}(\mathbf{s}_t, \mathbf{u}) + \gamma^{\Delta t \cdot v_{pref}} V(\hat{\mathbf{s}}_{t+1,\mathbf{u}}^{jn}), \tag{4.9}$$

under the simple kinematic model, f.

However, the introduction of parameter Δt leads to a difficult trade-off. Due to the the approximation of the value function in a DNN, a sufficiently large Δt is required such that each propagated $\hat{\mathbf{s}}_{t+1,\mathbf{u}}^{jn}$ is far enough apart, which ensures $V(\hat{\mathbf{s}}_{t+1,\mathbf{u}}^{jn})$ is not dominated by numerical noise in the network. The implication of large Δt is that agents are assumed to follow a constant velocity for a significant amount of time, which neglects the effects of cooperation/reactions to an agent's decisions. As the number of agents in the environment increases, this constant velocity assumption is less likely to be valid. Agents do not actually reach their propagated states because of the multiagent interactions.

The impact of separately querying the value function and performing collision checking is illustrated in Fig. 4-1. In (a), a red agent aims to reach its goal (star), and a purple agent is traveling at 1 m/s in the $-y$-direction. Because CADRL's value



(a) 2-Agent Scenario (b) Time-to-Goal Estimate, $V(\hat{\mathbf{s}}^{jn}_{t+1,\mathbf{u}})$ from DNN (c) Collision Cost, $R_{col}(\mathbf{s}_t, \mathbf{u})$, const. velocity model (d) Objective in using $\pi_{CADRL}(\mathbf{s}^{jn}_t)$

Figure 4-1: Issue with checking collisions and state-value separately, as in Eq. (4.9). In (a), the red agent's goal is at the star, and the purple agent's current velocity is in the $-y$-direction. In (b), the CADRL algorithm propagates the other agent forward at its current velocity (dashed purple circle), then queries the DNN for candidate future states. The best action (green star) is one which cuts above the purple agent, which was learned correctly by the CADRL V-Learning procedure. However, the constant velocity model of other agents is also used for collision checking, causing penalties of $R_{col}(\mathbf{s}_t, \mathbf{u})$, shown in (c). CADRL's policy combines these terms (d), instead choosing to go straight (green star), which is a poor choice that ignores that a cooperative purple agent likely would adjust its own velocity as well. This fundamental issue of checking collisions and state-values separately is addressed in this work by learning a policy directly.

function only encodes time-to-goal information, (b) depicts that the DNN appropriately recommends that the red agent should cut above the purple agent. However, there is a second term in Eq. (4.9) to convert the value function into a policy. This second term, the collision cost, $R_{col}(\mathbf{s}_t, \mathbf{u})$, shown in (c), penalizes actions that move toward the other agent's predicted position (dashed circle). This model-based collision checking procedure requires an assumption about other agents' behaviors, which is difficult to define ahead of time; the prior work assumed a constant-velocity model. When the value and collision costs are combined to produce $\pi_{CADRL}(\mathbf{s}^{jn}_t)$, the resulting objective-maximizing action is for the red agent to go straight, which will avoid a collision but be inefficient for both agents. The challenge in defining a model for other agents' behaviors was a primary motivation for learning a value function; even with an accurate value function, this example demonstrates an additional cause of inefficient paths: an inaccurate model used in the collision checking procedure.

In addition to not capturing decision making behavior of other agents, our experiments suggest that Δt is a crucial parameter to ensure convergence while training

the DNNs in the previous algorithms. If Δt is set too small or large, the training does not converge. A value of $\Delta t = 1$ sec was experimentally determined to enable convergence, though this number does not have much theoretical rationale.

In summary, the challenges of converting a value function to a policy, choosing the Δt hyperparameter, and our observation that the learning stability suffered with more than 4 agents in the environment each motivate the use of a different RL framework.

Policy Learning Therefore, this work considers RL frameworks which generate a policy that an agent can execute directly, without any arbitrary assumptions about state transition dynamics. A recent actor-critic algorithm by Mnih et al. called Asynchronous Advantage Actor-Critic (A3C) [83] uses a single DNN to approximate both the value (critic) and policy (actor) functions, and is trained with two loss terms

$$f_v = (R_t - V(\mathbf{s}_t^{jn}))^2, \tag{4.10}$$

$$f_\pi = \log \pi(\mathbf{u}_t|\mathbf{s}_t^{jn})(R_t - V(\mathbf{s}_t^{jn})) + \beta \cdot H(\pi(\mathbf{s}_t^{jn})), \tag{4.11}$$

where Eq. (4.10) trains the network's value output to match the future discounted reward estimate, $R_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(\mathbf{s}_{t+k}^{jn})$, over the next k steps, just as in CADRL. For the policy output in Eq. (4.11), the first term penalizes actions which have high probability of occurring ($\log \pi$) that lead to a lower return than predicted by the value function ($R - V$), and the second term encourages exploration by penalizing π's entropy with tunable constant β.

In A3C, many threads of an agent interacting with an environment are simulated in parallel, and a policy is trained based on an intelligent fusion of all the agents' experiences. The algorithm was shown to learn a policy that achieves superhuman performance on many video games. We specifically use GA3C [84], a hybrid GPU/CPU implementation that efficiently queues training experiences and action predictions proposed by Babaeizadeh et al. Our work builds on open-source GA3C implementations [84, 85].

Other choices for RL policy training algorithms (e.g., PPO [86], TD3 [87]) are

architecturally similar to A3C. Thus, the challenges mentioned above (varying number of agents, assumptions about other agents' behaviors) would map to future work that considers employing other RL algorithms or techniques [88] in this domain.

4.2.4 Related Works using Learning

There are several concurrent and subsequent works which use learning to solve the collision avoidance problem, categorized as non-RL, RL, and agent-level RL approaches.

Non-RL-based approaches to the collision avoidance problem include imitation learning, inverse RL, and supervised learning of prediction models. Imitation learning approaches [73] learn a policy that mimics what a human pedestrian or human teleoperator [89] would do in the same state but require data from an expert. Inverse RL methods learn to estimate pedestrians' cost functions, then use the cost function to inform robot plans [69, 90], but require real pedestrian trajectory data. Other approaches learn to predict pedestrian paths, which improves the world model used by the planner [91], but decoupling the prediction and planning steps could lead to the freezing robot problem (Section 4.2.2). A key advantage of RL over these methods is the ability to explore the state space through self-play, in which experiences generated in a low-fidelity simulation environment can reduce the need for expensive, real-world data collection efforts.

Within RL-based approaches, a key difference arises in the state representation: sensor-level and agent-level. Sensor-level approaches learn to select actions directly from raw sensor readings (either 2D laserscans [72] or images [73]) with end-to-end training. This leads to a large state space ($\mathbb{R}^{w \times h \times c}$ for a camera with resolution $w \times h$ and c channels, e.g., $480 \times 360 \times 3 = 5184000$), which makes training challenging. a Convolutional Neural Network (CNN) is often used to extract low-dimensional features from this giant state space, but training such a feature extractor in simulation requires an accurate sensor simulation model. The sensor-level approach has the advantage that both static and dynamic obstacles (including walls) can be fed into the network with a single framework. In contrast, this work uses interpretable clustering, tracking, and multi-sensor fusion algorithms to extract an agent-level state repre-

sentation from raw sensor readings. Advantages include a much smaller state space ($\mathbb{R}^{9+5(n-1)}$) enabling faster learning convergence; a sensor-agnostic collision avoidance policy, enabling sensor upgrades without re-training; and increased introspection into decision making, so that decisions can be traced back to the sensing, clustering, tracking, or planning modules.

Within agent-level RL, a key challenge is that of representing a variable number of nearby agents in the environment at any timestep. Typical feedforward networks used to represent the complex decision making policy for collision avoidance require a pre-determined input size. The sensor-level methods do maintain a fixed size input (sensor resolution), but have the limitations mentioned above. Instead, our first work trained a 2-agent value network, and proposed a mini-max rule to scale up to n agents [19]. To account for multiagent interactions (instead of only pairwise), our next work defines a maximum number of agents that the network can handle, and pads the observation space if there are actually fewer agents in the environment [20]. However, this maximum number of agents is limited by the increased number of network parameters (and therefore training time) as more agents' states are added. This work uses a recurrent network to convert a sequence of agent states at a particular timestep into a fixed-size representation of the world state; that representation is fed into the input of a standard feedforward network.

There are also differences in the reward functions used in RL-based collision avoidance approaches. Generally, the non-zero feedback provided at each timestep by a dense reward function (e.g., [72]) makes learning easier, but reward shaping quickly becomes a difficult problem in itself. For example, balancing multiple objectives (proximity to goal, proximity to others) can introduce unexpected and undesired local minima in the reward function. On the other hand, sparse rewards are easy to specify but require a careful initialization/exploration procedure to ensure agents will receive *some* environment feedback to inform learning updates. This work mainly uses sparse reward (arrival at goal, collision) with smooth reward function decay in near-collision states to encourage a minimum separation distance between agents. Additional terms in the reward function are shown to reliably induce higher-level

preferences (social norms) in our previous work [20].

While learning-based methods have many potential advantages over model-based approaches, learning-based approaches typically lack the guarantees (e.g., avoiding deadlock, zero collisions) desired for safety-critical applications. A key challenge in establishing guarantees in multiagent collision avoidance is what to assume about the world (e.g., policies and dynamics of other agents). Unrealistic or overly conservative assumptions about the world invalidate the guarantees or unnecessarily degrade the algorithm's performance: striking this balance may be possible in some domains but is particularly challenging in pedestrian-rich environments. A survey of the active research area of Safe RL is found in [92].

4.3 Approach

4.3.1 GA3C-CADRL

Recall the RL training process seeks to find the optimal policy, $\pi : \left(\mathbf{s}_t, \tilde{\mathbf{S}}_t^o\right) \mapsto \mathbf{u}_t$, which maps from an agent's observation of the environment to a probability distribution across actions and executes the action with highest probability. We use a local coordinate frame (rotation-invariant) as in [19, 20] and separate the state of the world in two pieces: information about the agent itself, and everything else in the world. Information about the agent can be represented in a small, fixed number of variables. The world, on the other hand, can be full of any number of other objects or even completely empty. Specifically, there is one $\mathbf{s}$ vector about the agent itself and one $\tilde{\mathbf{s}}^o$ vector per other agent in the vicinity:

$$\mathbf{s} = [d_g,\ v_{pref},\ \psi,\ r] \tag{4.12}$$

$$\tilde{\mathbf{s}}^o = [\tilde{p}_x,\ \tilde{p}_y,\ \tilde{v}_x,\ \tilde{v}_y,\ \tilde{r},\ \tilde{d}_a,\ \tilde{r} + r]\ , \tag{4.13}$$

where $d_g = ||\mathbf{p}_g - \mathbf{p}||_2$ is the agent's distance to goal, and $\tilde{d}_a = ||\mathbf{p} - \tilde{\mathbf{p}}||_2$ is the distance to the other agent.

The agent's action space is composed of a speed and change in heading angle. It

is discretized into 11 actions: with a speed of v_{pref} there are 6 headings evenly spaced between $\pm\pi/6$, and for speeds of $\frac{1}{2}v_{pref}$ and 0 the heading choices are $[-\pi/6,\ 0,\ \pi/6]$. These actions are chosen to mimic real turning constraints of robotic vehicles.

This multiagent RL problem formulation is solved with GA3C in a process we call Hybrid GPU/CPU Asynchronous Advantage Actor-Critic (GA3C) CADRL (GA3C-CADRL). Since experience generation is one of the time-intensive parts of training, this work extends GA3C to learn from multiple agents' experiences each episode. Training batches are filled with a mix of agents' experiences ($\{\mathbf{s}_t^{jn},\ \mathbf{u}_t,\ r_t\}$ tuples) to encourage policy gradients that improve the joint expected reward of all agents. Our multiagent implementation of GA3C accounts for agents reaching their goals at different times and ignores experiences of agents running other policies (e.g., non-cooperative agents).

4.3.2 Handling a Variable Number of Agents

Recall that one key limitation of many learning-based collision avoidance methods is that the feedforward DNNs typically used require a fixed-size input. Convolutional and max-pooling layers are useful for feature extraction and can modify the input size but still convert a fixed-size input into a fixed-size output. Recurrent DNNs, where the output is produced from a combination of a stored cell state and an input, accept an arbitrary-length sequence to produce a fixed-size output. Long short-term memory (LSTM) [76] is recurrent architecture with advantageous properties for training[4].

Although LSTMs are often applied to time sequence data (e.g., pedestrian motion prediction [93]), this paper leverages their ability to encode a sequence of information that is not time-dependent (see [94] for a thorough explanation of LSTM calculations). LSTM is parameterized by its weights, $\{W_i, W_f, W_o\}$, and biases, $\{b_i, b_f, b_o\}$, where $\{i, f, o\}$ correspond to the input, forget, and output gates. The variable number of $\tilde{\mathbf{s}}_i^o$ vectors is a sequence of inputs that encompass everything the agent knows about the rest of the world. As depicted in Fig. 4-2, each LSTM *cell* has three inputs: the state

[4]In practice, TensorFlow's LSTM implementation requires a known maximum sequence length, but this can be set to something bigger than the number of agents agents ever expected (e.g., 20)

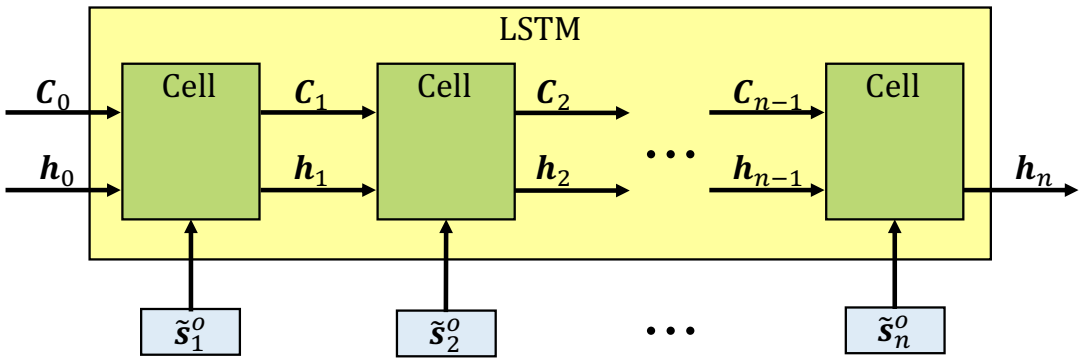


Figure 4-2: LSTM unrolled to show each input. At each decision step, the agent feeds one observable state vector, $\tilde{\mathbf{s}}_i^o$, for each nearby agent, into a LSTM cell sequentially. LSTM cells store the pertinent information in the hidden states, $\mathbf{h}_i$. The final hidden state, $\mathbf{h}_n$, encodes the entire state of the other agents in a fixed-length vector, and is then fed to the feedforward portion of the network. The order of agents is sorted by decreasing distance to the ego agent, so that the closest agent has the most recent effect on $\mathbf{h}_n$.

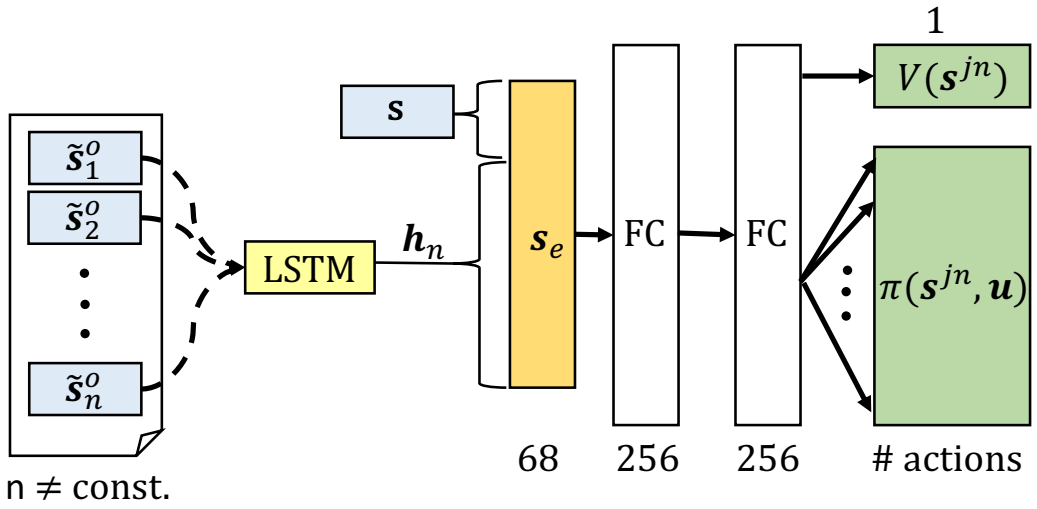


Figure 4-3: Network Architecture. Observable states of nearby agents, $\tilde{\mathbf{s}}_i^o$, are fed sequentially into the LSTM, as unrolled in Fig. 4-2. The final hidden state is concatenated with the agent's own state, $\mathbf{s}$, to form the vector, $\mathbf{s}_e$. For any number of agents, $\mathbf{s}_e$ contains the agent's knowledge of its own state and the state of the environment. The encoded state is fed into two fully-connected layers (FC). The outputs are a scalar value function (top, right) and policy represented as a discrete probability distribution over actions (bottom, right).

of agent j at time t, the previous hidden state, and the previous cell state, which are denoted $\tilde{\mathbf{s}}^o_{j,t}$, $\mathbf{h}_j$, C_j, respectively. Thus, at each decision step, the agent feeds each $\tilde{\mathbf{s}}^o_i$ (observation of i^{th} other agent's state) into a LSTM cell sequentially. That is, the LSTM initially has empty states ($\mathbf{h}_0, \mathbf{C}_0$ set to zeros) and uses $\{\tilde{\mathbf{s}}^o_1, \mathbf{h}_0, C_0\}$ to generate $\{\mathbf{h}_1, C_1\}$, then feeds $\{\tilde{\mathbf{s}}^o_2, \mathbf{h}_1, C_1\}$ to produce $\{\mathbf{h}_2, C_2\}$, and so on. As agents' states are fed in, the LSTM "remembers" the pertinent information in its hidden/cell states, and "forgets" the less important parts of the input (where the notion of memory is parameterized by the trainable LSTM weights/biases). After inputting the final agent's state, we can interpret the LSTM's final hidden state, $\mathbf{h}_n$ as a fixed-length, encoded state of the world, for that decision step. The LSTM contains n cells, so the entire module receives inputs $\{\tilde{\mathbf{S}}^o_t, \mathbf{h}_{t-1}, \mathbf{C}_{t-1}\}$ and produces outputs $\{\mathbf{h}_n, \mathbf{C}_n\}$, and $\mathbf{h}_n$ is passed to the next network layer for decision making.

Given a sufficiently large hidden state vector, there is enough space to encode a large number of agents' states without the LSTM having to forget anything relevant. In the case of a large number of agent states, to mitigate the impact of the agent forgetting the early states, the states are fed in reverse order of distance to the agent, meaning the closest agents (fed last) should have the biggest effect on the final hidden state, $\mathbf{h}_n$. Because the list of agents needs to be ordered in some manner, reverse distance is one possible ordering heuristic – we empirically compare to other possibilities in Section 4.4.4.

Another interpretation of the LSTM objective is that it must learn to combine an observation of a new agent with a representation of other agents (as opposed to the architectural objective of producing a fixed-length encoding of a varying size input). This interpretation provides intuition on how an LSTM trained in 4-agent scenarios can generalize reasonably well to cases with 10 agents.

The addition of LSTM to a standard actor-critic network is visualized in Fig. 4-3, where the box labeled $\mathbf{s}$ is the agent's own state, and the group of boxes is the n other agents' observable states, $\tilde{\mathbf{s}}^o_i$. After passing the n other agents' observable states into the LSTM, the agent's own state is concatenated with $\mathbf{h}_n$ to produce the encoded representation of the joint world state, $\mathbf{s}_e$. Then, $\mathbf{s}_e$ is passed to a typical feedforward

DNN with 2 fully-connected layers (256 hidden units each with ReLU activation).

The network produces two output types: a scalar state value (critic) and a policy composed of a probability for each action in the discrete action space (actor). During training, the policy *and* value are used for Eqs. (4.10) and (4.11); during execution, only the policy is used. During the training process, the LSTM's weights are updated to learn how to represent the variable number of other agents in a fixed-length **h** vector. The whole network is trained end-to-end with backpropagation.

4.3.3 Training the Policy

The original Collision Avoidance with Deep Reinforcement Learning (CADRL) and Socially Aware CADRL (SA-CADRL) algorithms used several clever tricks to enable convergence when training the networks. Specifically, forward propagation of other agent states for Δt seconds was a critical component that required tuning, but does not represent agents' true behaviors. Other details include separating experiences into successful/unsuccessful sets to focus the training on cases where the agent could improve. The new GA3C-CADRL formulation is more general, and does not require such assumptions or modifications.

The training algorithm is described in Algorithm 3. In this work, to train the model, the network weights are first initialized in a supervised learning phase, which converges in less than five minutes. The initial training is done on a large, publicly released set of state-action-value tuples, $\{\mathbf{s}_t^{jn}, \mathbf{u}_t, V(\mathbf{s}_t^{jn}; \phi_{CADRL})\}$, from an existing CADRL solution. The network loss combines square-error loss on the value output and softmax cross-entropy loss between the policy output and the one-hot encoding of the closest discrete action to the one in D, described in Lines 3-8 of Algorithm 3.

The initialization step is necessary to enable any possibility of later generating useful RL experiences (non-initialized agents wander randomly and probabilistically almost never obtain positive reward). Agents running the initialized GA3C-CADRL policy reach their goals reliably when there are no interactions with other agents. However, the policy after this supervised learning process still performs poorly in collision avoidance. This observation contrasts with CADRL, in which the initialization

step was sufficient to learn a policy that performs comparably to existing reaction-based methods, due to relatively-low dimension value function combined with manual propagation of states. Key reasons behind this contrast are the reduced structure in the GA3C-CADRL formulation (no forward propagation), and that the algorithm is now learning both a policy and value function (as opposed to just a value function), since the policy has an order of magnitude higher dimensionality than a scalar value function.

To improve the solution with RL, parallel simulation environments produce training experiences, described in Lines 9-22 of Algorithm 3. Each episode consists of 2-10 agents, with random start and goal positions, running a random assortment of policies (Non-Cooperative, Zero Velocity, or the learned GA3C-CADRL policy at that iteration) (Line 10). Agent parameters vary between $r \in [0.2, 0.8]$m and $v_{pref} \in [0.5, 2.0]$m/s, chosen to be near pedestrian values. Agents sense the environment and transform measurements to their ego frame to produce the observation vector (Lines 13, 14). Each agent sends its observation vector to the policy queue and receives an action sampled from the current iteration of the GA3C-CADRL policy (Line 15). Agents that are not running the GA3C-CADRL policy use their own policy to overwrite $\mathbf{u}_{t,j}$ (Line 18). Then, all agents that have not reached a terminal condition (collision, at goal, timed out), simultaneously move according to $\mathbf{u}_{t,j}$ (Line 19). After all agents have moved, the environment evaluates $R(\mathbf{s}^{jn}, \mathbf{u})$ for each agent, and experiences from GA3C-CADRL agents are sent to the training queue (Lines 21,22).

In another thread, experiences are popped from the queue to produce training batches (Line 24). These experience batches are used to train a single GA3C-CADRL policy (Line 25) as in [84].

An important benefit of the new framework is that the policy can be trained on scenarios involving any number of agents, whereas the maximum number of agents had to be defined ahead of time with CADRL/SA-CADRL[5]. This work begins the RL phase with 2-4 agents in the environment, so that the policy learns the idea of

[5]Experiments suggest this number should be below about 6 for convergence

Algorithm 3: GA3C-CADRL Training

Input: trajectory training set, D
Output: policy network $\pi(\cdot;\theta)$
// Initialization
for N_{epochs} **do**
  $\{\mathbf{s}_t^o, \tilde{\mathbf{S}}_t^o, \mathbf{u}_t, V_t\} \leftarrow \text{grabBatch}(D)$
  $\bar{\mathbf{u}}_t \leftarrow \text{closestOneHot}(\mathbf{u}_t)$
  $\mathcal{L}_V = (V_t - V(\mathbf{s}_t^o, \tilde{\mathbf{S}}_t^o; \phi)^2$
  $\mathcal{L}_\pi = \text{softmaxCELogits}(\bar{\mathbf{u}}_t, \mathbf{s}_t^o, \tilde{\mathbf{S}}_t^o, \theta)$
  $\pi(\cdot;\theta), V(\cdot;\phi) \leftarrow \text{trainNNs}(\mathcal{L}_\pi, \mathcal{L}_V, \theta, \phi)$
// Parallel Environment Threads
foreach *env* **do**
  $\mathbf{S}_0 \leftarrow \text{randomTestCase}()$
  while *some agent not done* **do**
    foreach *agent, j* **do**
      $\mathbf{s}_g^o, \tilde{\mathbf{S}}_g^o \leftarrow \text{sensorUpdate}()$
      $\mathbf{s}^o, \tilde{\mathbf{S}}^o \leftarrow \text{transform}(\mathbf{s}_g^o, \tilde{\mathbf{S}}_g^o)$
    $\{\mathbf{u}_{t,j}\} \sim \pi(\mathbf{s}_{t,j}^o, \tilde{\mathbf{S}}_{t,j}^o; \theta) \forall j$
    foreach *not done agent, j* **do**
      if *agent not running GA3C-CADRL* **then**
        $\mathbf{u}_{t,j} \leftarrow \text{policy}(\mathbf{s}_{t,j}^o, \tilde{\mathbf{S}}_{t,j}^o)$
      $\mathbf{s}_{j,t+1}, \tilde{\mathbf{S}}_{j,t+1}, r_{j,t} \leftarrow \text{moveAgent}(\mathbf{u}_{j,t})$
    foreach *not done GA3C-CADRL agent, j* **do**
      $r_{t,j} \leftarrow \text{checkRewards}(\mathbf{S}_{t+1}, \mathbf{u}_{t,j})$
      $\text{addToExperienceQueue}(\mathbf{s}_{t,j}^o, \tilde{\mathbf{S}}_{t,j}^o, \mathbf{u}_{t,j}, r_{t,j})$
// Training Thread
for $N_{episodes}$ **do**
  $\{\mathbf{s}_{t+1}^o, \tilde{\mathbf{S}}_{t+1}^o, \mathbf{u}_t, r_t\} \leftarrow \text{grabBatchFromQueue}()$
  $\theta, \phi \leftarrow \text{trainGA3C}(\theta, \phi, \{\mathbf{s}_{t+1}^o, \tilde{\mathbf{S}}_{t+1}^o, \mathbf{u}_t, r_t\})$
return π

Algorithm 4: GA3C-CADRL Execution

1 **Input:** goal position, (g_x, g_y)
2 **Output:** next motor commands, $\mathbf{u}$
3 $\mathbf{s}_g^o, \tilde{\mathbf{S}}_g^o \leftarrow$ sensorUpdate()
4 $\mathbf{s}^o, \tilde{\mathbf{S}}^o \leftarrow \text{transform}(\mathbf{s}_g^o, \tilde{\mathbf{S}}_g^o)$
5 $s_{des}, \theta_{des} \leftarrow \pi(\mathbf{s}^o, \tilde{\mathbf{S}}^o)$
6 $\mathbf{u} \leftarrow \text{control}(s_{des}, \theta_{des})$
7 **return u**

collision avoidance in reasonably simple domains. Upon convergence, a second RL phase begins with 2-10 agents in the environment.

4.3.4 Policy Inference

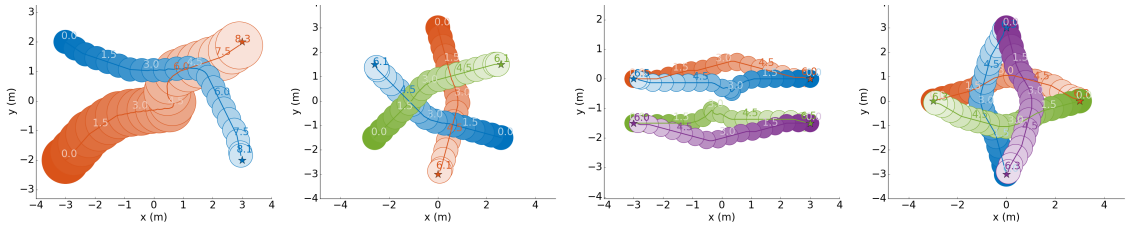
Inference of the trained policy for a single timestep is described in Algorithm 4. As in training, GA3C-CADRL agents sense the environment, transfer to the ego frame, and select an action according to the policy (Lines 3-5). Like many RL algorithms, actions are sampled from the stochastic policy during training (exploration), but the action with highest probability mass is selected during inference (exploitation). A necessary addition for hardware is a low-level controller to track the desired speed and heading angle (Line 6). Note that the value function is not used during inference; it is only learned to stabilize estimates of the policy gradients during training.

4.4 Results

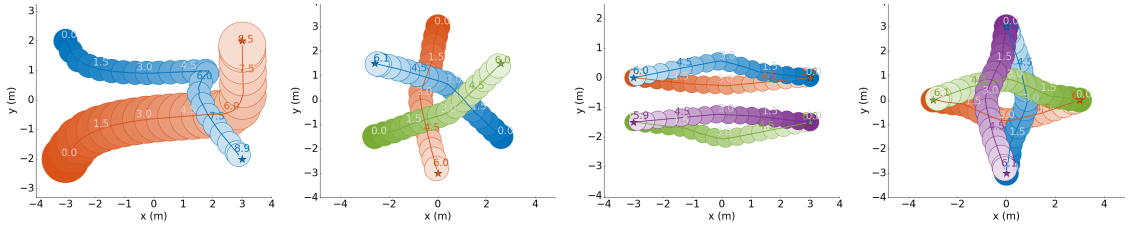
4.4.1 Computational Details

The DNNs in this work were implemented with TensorFlow [95] in Python. Each query of the GA3C-CADRL network only requires the current state vector, and takes on average 0.4-0.5ms on a i7-6700K CPU, which is approximately 20 times faster than before [20]. Note that a GPU is not required for online inference in real time, and CPU-only training was faster than hybrid CPU-GPU training on our hardware.

In total, the RL training converges in about 24 hours (after $2 \cdot 10^6$ episodes)



(a) GA3C-CADRL trajectories with $n \in [2, 3, 4]$ agents

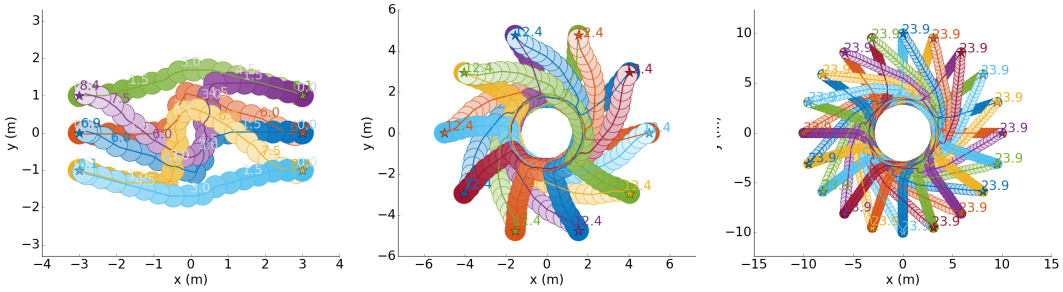


(b) SA-CADRL trajectories with $n \in [2, 3, 4]$ agents

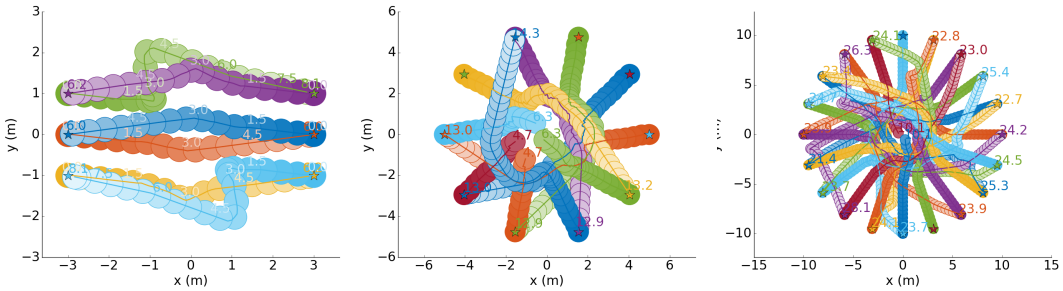
Figure 4-4: Scenarios with $n \leq 4$ agents. The top row shows agents executing GA3C-CADRL-10-LSTM, and the bottom row shows same scenarios with agents using SA-CADRL. Circles lighten as time increases, the numbers represent the time at agent's position, and circle size represents agent radius. GA3C-CADRL agents are slightly less efficient, as they reach their goals slightly slower than SA-CADRL agents. However, the overall behavior is similar, and the more general GA3C-CADRL framework generates desirable behavior without many of the assumptions from SA-CADRL.

for the multiagent, LSTM network on a computer with an i7-6700K CPU with 32 parallel environment threads. A limiting factor of the training time is the low learning rate required for stable training. Recall that earlier approaches [20] took 8 hours to train a 4-agent value network, but now the network learns both the policy and value function and without being provided any structure about the other agents' behaviors. The larger number of training episodes can also be attributed to the stark contrast in initial policies upon starting RL between this and the earlier approach: CADRL was fine-tuning a decent policy, whereas GA3C-CADRL learns collision avoidance entirely in the RL phase.

The performance throughout the training procedure is shown as the "closest last" curve in Fig. 4-7 (the other curves are explained in Section 4.4.4). The mean $\pm 1\sigma$ rolling reward over 5 training runs is shown. After initialization, the agents receive on average 0.15 reward per episode. After RL phase 1 (converges in $1.5 \cdot 10^6$ episodes), they average 0.90 rolling reward per episode. When RL phase 2 begins, the domain

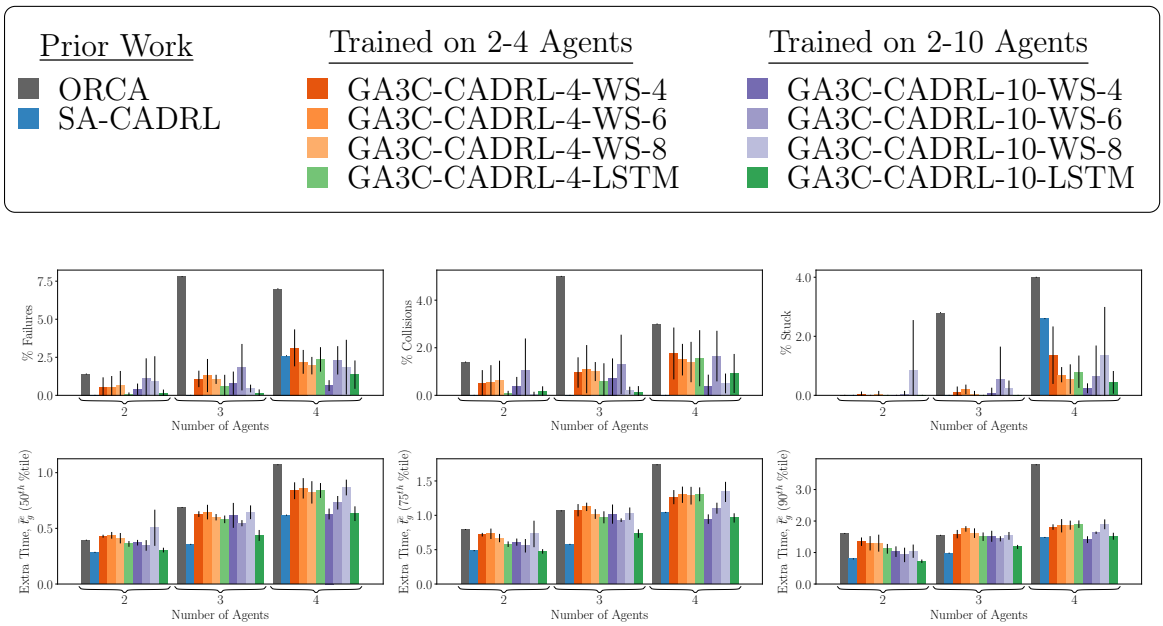


(a) GA3C-CADRL: 3-pair swaps (b) GA3C-CADRL: 10-agent circle (c) GA3C-CADRL: 20-agent circle

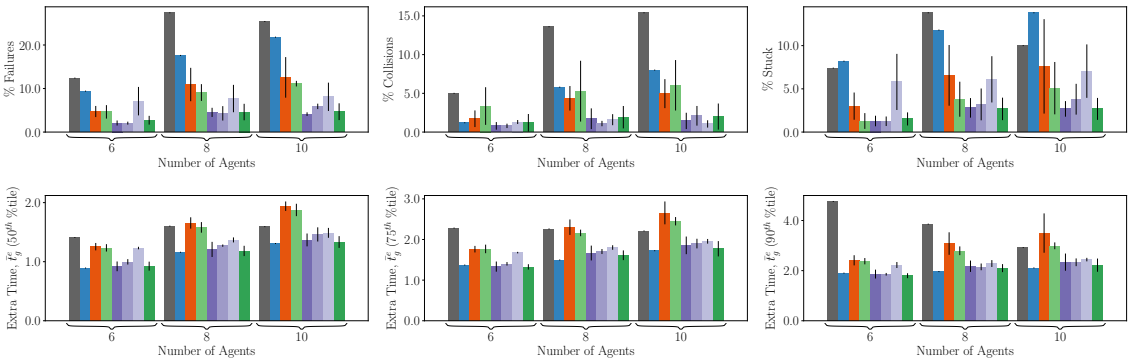


(d) SA-CADRL: 3-pair swaps (e) SA-CADRL: 10-agent circle (f) SA-CADRL: 20-agent circle

Figure 4-5: Scenarios with $n > 4$ agents. In the 3-pair swap Figs. 4-5a and 4-5d, GA3C-CADRL agents exhibit interesting multiagent behavior: two agents form a pair while passing the opposite pair of agents. SA-CADRL agents reach the goal more quickly than GA3C-CADRL agents, but such multiagent behavior is a result of GA3C-CADRL agents having the capacity to observe all of the other 5 agents each time step. In other scenarios, GA3C-CADRL agents successfully navigate the 10- and 20-agent circles, whereas some SA-CADRL agents collide (near $(-1, -1)$ and $(0, 0)$ in Fig. 4-5e and $(0, 0)$ in Fig. 4-5f).



(a) Comparison of Algorithms for $n \leq 4$ agents



(b) Comparison of Algorithms for $n > 4$ agents

Figure 4-6: Numerical comparison on the same 500 random test cases (lower is better). The GA3C-CADRL-10-LSTM network shows comparable performance to SA-CADRL for small n (a), much better performance for large n (b), and better performance than model-based ORCA for all n. Several ablations highlight SA-CADRL and GA3C-CADRL differences. With the same architecture (SA-CADRL & GA3C-CADRL-4-WS-4), the GA3C policy performs better for large n (b), but worsens performance for small n (a). Adding a second phase of training with up to 10 agents (GA3C-CADRL-10-WS-4) improves performance for all n tested. Adding additional pre-defined agent capacity to the network (GA3C-CADRL-10-WS-6, GA3C-CADRL-10-WS-8) can degrade performance. The LSTM (GA3C-CADRL-10-LSTM) adds flexibility in prior knowledge on number of agents, maintaining similar performance to the WS approaches for large n and recovering comparable performance to SA-CADRL for small n.

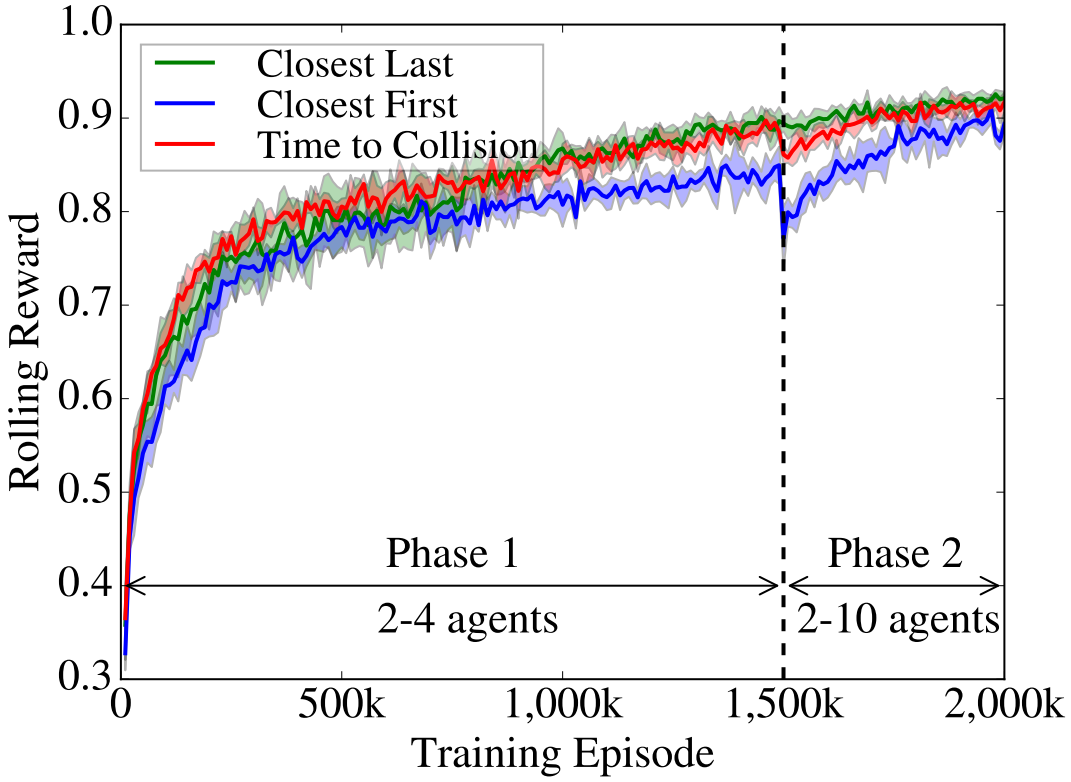


Figure 4-7: Training performance and LSTM ordering effect on training. The first phase of training uses random scenarios with 2-4 agents; the final 500k episodes use random scenarios with 2-10 agents. Three curves corresponding to three heuristics for ordering the agent sequences all converge to a similar reward after 2M episodes. The "closest last" ordering has almost no dropoff between phases and achieves the highest final performance. The "closest first" ordering drops off substantially between phases, suggesting the ordering scheme has a second-order effect on training. Curves show the mean $\pm 1\sigma$ over 5 training runs per ordering.

becomes much harder (n_{max} increases from 4 to 10), and rolling reward increases until converging at 0.93 (after a total of $1.9 \cdot 10^6$ episodes). Rolling reward is computed as the sum of the rewards accumulated in each episode, averaged across all GA3C-CADRL agents in that episode, averaged over a window of recent episodes. Rolling reward is only a measure of success/failure, as it does not include the discount factor and thus is not indicative of time efficiency. Because the maximum receivable reward on any episode is 1, an average reward < 1 implies there are some collisions (or other penalized behavior) even after convergence. This is expected, as agents sample from their policy distributions when selecting actions in training, so there is always a non-

zero probability of choosing a sub-optimal action in training. Later, when executing a trained policy, agents select the action with highest probability.

Key hyperparameter values include: learning rate $L_r = 2 \cdot 10^{-5}$, entropy coefficient $\beta = 1 \cdot 10^{-4}$, discount $\gamma = 0.97$, training batch size $b_s = 100$, and we use the Adam optimizer [96].

4.4.2 Simulation Results

Baselines

This section compares the proposed GA3C-CADRL algorithm to:

- Optimal Reciprocal Collision Avoidance (ORCA) [67],
- Socially Aware CADRL (SA-CADRL) [20], and
- Deep Reinforcement Learning Multiagent Collision Avoidance (DRLMACA) [72] (where applicable).

In our experiments, ORCA agents must inflate agent radii by 5% to reduce collisions caused by numerical issues. Without this inflation, over 50% of experiments with 10 ORCA agents had a collision. This inflation led to more ORCA agents getting stuck, which is better than a collision in most applications. The time horizon parameter in ORCA impacts the trajectories significantly; it was set to 5 seconds.

Although the original 2-agent CADRL algorithm [19] was also shown to scale to multiagent scenarios, its minimax implementation is limited in that it only considers one neighbor at a time as described in [20]. For that reason, this work focuses on the comparison against SA-CADRL which has better multiagent properties - the policy used for comparison is the same one that was used on the robotic hardware in [20]. That particular policy was trained with some noise in the environment ($\mathbf{p} = \mathbf{p}_{actual} + \sigma$) which led to slightly poorer performance than the ideally-trained network as reported in the results of [20], but more acceptable hardware performance.

The version of the new GA3C-CADRL policy after RL phase 2 is denoted GA3C-CADRL-10, as it was trained in scenarios of up to 10 agents. To create a more fair comparison with SA-CADRL which was only trained with up to 4 agents, let GA3C-

CADRL-4 denote the policy after RL phase 1 (which only involves scenarios of up to 4 agents). Recall GA3C-CADRL-4 can still be naturally implemented on $n > 4$ agent cases, whereas SA-CADRL can only accept up to 3 nearby agents' states regardless of n.

$n \leq 4$ agents: Numerical Comparison to Baselines

The previous approach (SA-CADRL) is known to perform well on scenarios involving a few agents ($n \leq 4$), as its trained network can accept up to 3 other agents' states as input. Therefore, a first objective is to confirm that the new algorithm can still perform comparably with small numbers of agents. This is not a trivial check, as the new algorithm is not provided with any structure/prior about the world's dynamics, so the learning is more difficult.

Trajectories are visualized in Fig. 4-4: the top row shows scenarios with agents running the new policy (GA3C-CADRL-10-LSTM), and the bottom row shows agents in identical scenarios but using the old policy (SA-CADRL). The colors of the circles (agents) lighten as time increases and the circle size represents agent radius. The trajectories generally look similar for both algorithms, with SA-CADRL being slightly more efficient. A rough way to assess efficiency in these plotted paths is time indicated when the agents reach their goals.

Although it is easy to pick out interesting pros/cons for any particular scenario, it is more useful to draw conclusions after aggregating over a large number of randomly-generated cases. Thus, we created test sets of 500 random scenarios, defined by (p_{start}, p_{goal}, r, v_{pref}) per agent, for many different numbers of agents. Each algorithm is evaluated on the same 500 test cases. The comparison metrics are the percent of cases with a collision, percent of cases where an agent gets stuck and doesn't reach the goal, and the remaining cases where the algorithm was successful, the average extra time to goal, $\bar{t}_g^e$ beyond a straight path at v_{pref}. These metrics provide measures of efficiency and safety.

Aggregated results in Fig. 4-6 compare a model-based algorithm, ORCA [67], SA-CADRL [20], and several variants of the new GA3C-CADRL algorithm. With $n \leq 4$

agents in the environment (a), SA-CADRL has the lowest $\bar{t}_g^e$, and the agents rarely fail in these relatively simple scenarios.

$n \leq 4$ agents: Ablation Study

There are several algorithmic differences between SA-CADRL and GA3C-CADRL: we compare each ablation one-by-one. With the same network architecture (pre-defined number of agents with weights shared (WS) for all agents), GA3C-CADRL-4-WS-4 loses some performance versus SA-CADRL, since GA3C-CADRL must learn the notion of a collision, whereas SA-CADRL's constant-velocity collision checking may be reasonable for small n. Replacing the WS network head with LSTM improves the performance when the number of agents is below network capacity, potentially because the LSTM eliminates the need to pass "dummy" states to fill the network input vector. Lastly, the second training phase (2-10 agents) improves policy performance even for small numbers of agents.

Overall, the GA3C-CADRL-10-LSTM variant performs comparably, though slightly worse, than SA-CADRL for small numbers of agents, and outperforms the model-based ORCA algorithm.

$n > 4$ agents: Numerical Comparison to Baselines

A real robot will likely encounter more than 3 pedestrians at a time in a busy environment. However, experiments with the SA-CADRL algorithm suggest that increasing the network capacity beyond 4 total agents causes convergence issues. Thus, the approach taken here for SA-CADRL is to supply only the closest 3 agents' states in crowded scenarios. The GA3C-CADRL policy's convergence is not as sensitive to the maximum numbers of agents, allowing an evaluation of whether simply expanding the network input size improves performance in crowded scenarios. Moreover, the LSTM variant of GA3C-CADRL relaxes the need to pre-define a maximum number of agents, as any number of agents can be fed into the LSTM and the final hidden state can still be taken as a representation of the world configuration.

Even in $n > 4$-agent environments, interactions still often only involve a couple of

agents at a time. Some specific cases where there truly are many-agent interactions are visualized in Fig. 4-5. In the 6-agent swap (left), GA3C-CADRL agents exhibit interesting multiagent behavior: the bottom-left and middle-left agents form a pair while passing the top-right and middle-right agents. This phenomenon leads to a particularly long path for bottom-left and top-right agents, but also allows the top-left and bottom-right agents to not deviate much from a straight line. In contrast, in SA-CADRL the top-left agent starts moving right and downward, until the middle-right agent becomes one of the closest 3 neighbors. The top-left agent then makes an escape maneuver and passes the top-right on the outside. In this case, SA-CADRL agents reach the goal more quickly than GA3C-CADRL agents, but the interesting multiagent behavior is a result of GA3C-CADRL agents having the capacity to observe all of the other 5 agents each time step, rather than SA-CADRL which just uses the nearest 3 neighbors. GA3C-CADRL agents successfully navigate the 10- and 20-agent circles (antipodal swaps), whereas several SA-CADRL agents get stuck or collide[6].

Statistics across 500 random cases of 6, 8, and 10 agents are shown in Fig. 4-6b. The performance gain by using GA3C-CADRL becomes larger as the number of agents in the environment increases. For $n = 6, 8, 10$, GA3C-CADRL-10-LSTM shows a 3-4x reduction in failed cases with similar $\bar{t}_g^e$ compared to SA-CADRL. GA3C-CADRL-10-LSTM's percent of success remains above 95% across any $n \leq 10$, whereas SA-CADRL drops to under 80%. It is worth noting that SA-CADRL agents' failures are more often a result of getting stuck rather than colliding with others, however neither outcomes are desirable. The GA3C-CADRL variants outperform model-based ORCA for large n as well. The domain size of $n = 10$ agent scenarios is set to be larger (12×12 vs. $8 \times 8m$) than cases with smaller n to demonstrate cases where n is large but the world is not necessarily more densely populated with agents.

[6]Note there is not perfect symmetry in these SA-CADRL cases: small numerical fluctuations affect the choice of the closest agents, leading to slightly different actions for each agent. And after a collision occurs with a pair of agents, symmetry will certainly be broken for future time steps.

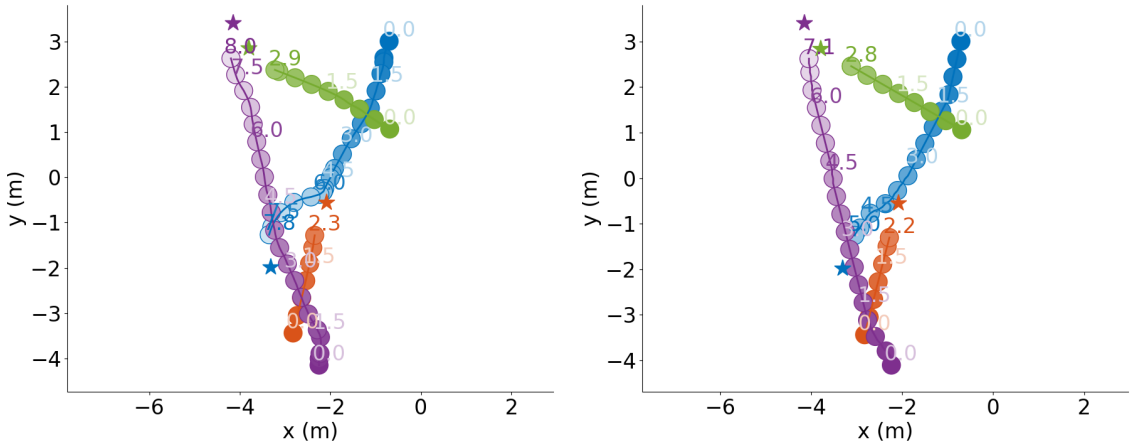
$n > 4$ agents: Ablation Study

We now discuss the GA3C-CADRL variants. For large n, GA3C-CADRL-10-WS-4 strongly outperforms SA-CADRL. Since the network architectures and number of agents trained on are the same, the performance difference highlights the benefit of the policy-based learning framework. Particularly for large n, the multiagent interactions cause SA-CADRL's constant velocity assumption about other agents (to convert the value function to a policy in Eq. (4.9)) to become unrealistic. Replacing the WS network head with LSTM (which accepts observations of any number of agents) causes slight performance improvement for $n = 6, 8$ (GA3C-CADRL-4-WS-4 vs. GA3C-CADRL-4-LSTM). Since GA3C-CADRL-4-WS-6,8 never saw $n > 4$ agents in training, these are omitted from Fig. 4-6b (as expected, their performance is awful). The second training phase (on up to 10 agents) leads to another big performance improvement (GA3C-CADRL-4-* vs. GA3C-CADRL-10-*). The additional network capacity of the WS approaches (GA3C-CADRL-10-WS-4,6,8) appears to have some small, in some cases negative, performance impact. That observation suggests that simply increasing the maximum number of agents input to the network does not address the core issues with multiagent collision avoidance. The GA3C-CADRL-10-LSTM variant performs similarly to GA3C-CADRL-10-WS-4, while providing a more flexible network architecture (and better performance for small n, as described earlier).

The ability for GA3C-CADRL to retrain in complex scenarios after convergence in simple scenarios, and yield a significant performance increase, is a key benefit of the new framework. This result suggests there could be other types of complexities in the environment (beyond increasing n) that the general GA3C-CADRL framework could also learn about after being initially trained on simple scenarios.

Comparison to Other RL Approach

Table 4.1 shows a comparison to another deep RL policy, DRLMACA [72]. DRLMACA stacks the latest 3 laserscans as the observation of other agents; other al-



(a) DRLMACA trajectory with 4 agents (b) GA3C-CADRL trajectory with 4 agents

Figure 4-8: GA3C-CADRL and DRLMACA 4-agent trajectories. Both algorithms produce collision-free paths; numbers correspond to the timestamp agents achieved that position. GA3C-CADRL agents are slightly faster; the bottom-most DRLMACA agent (left) slows down near the start of the trajectory, leading to a larger time to goal (8.0 vs. 7.1 seconds), whereas the bottom-most GA3C-CADRL agent (right) cuts behind another agent near its v_{pref}. Similarly, the top-right DRLMACA agent slows down near $(-2, 0)$ (overlapping circles), whereas the top-right GA3C-CADRL agent maintains high speed and reaches the goal faster (7.8 vs. 5.0 seconds). Agents stop moving once within $0.8m$ of their goal position in these comparisons.

gorithms in the comparisons use the exact other agent states. DRLMACA assumes $v_{pref} = 1m/s$ for all agents, so all test cases used in Table 4.1 share this setting (v_{pref} is random in Fig. 4-6, explaining the omission of DRLMACA).

During training, all DRLMACA agents are discs of the same radius, R, and some reported trajectories from [72] suggest the policy can generalize to other agent sizes. However, our experiments with a trained DRLMACA policy [97] suggest the policy does not generalize to other agent radii, as the number of collisions increases with agent radius. In particular, 69% of experiments ended in a collision for 4 agents running DRLMACA with $r = 0.5m$. Moreover, a qualitative look at DRLMACA trajectories in Fig. 4-8 demonstrates how agents often slow down and stop to wait for other agents, whereas GA3C-CADRL agents often move out of other agents' paths before needing to stop. Even though the implementation in [97] uses the same hyperparameters, training scheme, network architecture, and reward function from the paper, these results are worse than what was reported in [72].

Table 4.1: Performance of ORCA [67], SA-CADRL [20], DRLMACA [72], and GA3C-CADRL (new) algorithms on the same 100 random test cases, for various agent radii, with $v_{pref} = 1.0m/s$ for all agents. For both $r = 0.2m$ and $r = 0.5m$, GA3C-CADRL outperforms DRLMACA, and DRLMACA performance drops heavily for $r = 0.5m$, with 69% collisions in random 4-agent scenarios.

	Test Case Setup		
size (m)		8 x 8	8 x 8
# agents		2	4
	Extra time to goal $\bar{t}_g^e$ (s) (Avg / 75th / 90th percentile) ⇒ smaller is better		
ORCA	$r = 0.2m$	0.18 / 0.39 / 0.82	0.4 / 0.62 / 2.4
SA-CADRL		0.2 / 0.29 / 0.43	**0.26 / 0.38 / 0.75**
DRLMACA		0.91 / 1.33 / 4.82	1.46 / 2.16 / 3.35
GA3C-CADRL-10-LSTM		0.17 / 0.24 / 0.71	**0.25 / 0.53 / 0.7**
	% failures (% collisions / % stuck) ⇒ smaller is better		
ORCA	$r = 0.2m$	4 (4 / 0)	7 (7 / 0)
SA-CADRL		**0 (0 / 0)**	2 (0 / 2)
DRLMACA		3 (0 / 3)	14 (8 / 6)
GA3C-CADRL-10-LSTM		**0 (0 / 0)**	**0 (0 / 0)**
	Extra time to goal $\bar{t}_g^e$ (s) (Avg / 75th / 90th percentile) ⇒ smaller is better		
ORCA	$r = 0.5m$	0.43 / 1.11 / 1.65	0.95 / 1.22 / 1.86
SA-CADRL		**0.27 / 0.37 / 0.66**	**0.48 / 1.03 / 1.71**
DRLMACA		0.72 / 1.12 / 1.33	1.26 / 2.42 / 2.57
GA3C-CADRL-10-LSTM		**0.27 / 0.37 / 0.57**	**0.6 / 1.24 / 1.69**
	% failures (% collisions / % stuck) ⇒ smaller is better		
ORCA	$r = 0.5m$	4 (4 / 0)	12 (9 / 3)
SA-CADRL		**0 (0 / 0)**	2 (0 / 2)
DRLMACA		23 (23 / 0)	71 (69 / 2)
GA3C-CADRL-10-LSTM		**0 (0 / 0)**	**1 (1 / 0)**

Formation Control

Formation control is one application of multiagent robotics that requires collision avoidance: recent examples include drone light shows [98], commercial airplane formations [99], robotic soccer [100], and animations [101]. One possible formation objective is to assign a team of agents to goal coordinates, say to spell out letters or make a shape.

Fig. 4-9 shows 6 agents spelling out the letters in "CADRL". Each agent uses GA3C-CADRL-10-LSTM and knowledge of other agents' current positions, velocities, and radii, to choose a collision-free action toward its own goal position. All goal coordinates lie within a $6 \times 6m$ region, and goal coordinates are randomly assigned to agents. Each agent has a radius of $0.5m$ and a preferred speed of $1.0m/s$. Agents start in random positions before the first letter, "C", then move from "C" to "A", etc. Agent trajectories darken as time increases, and the circles show the final agent positions.

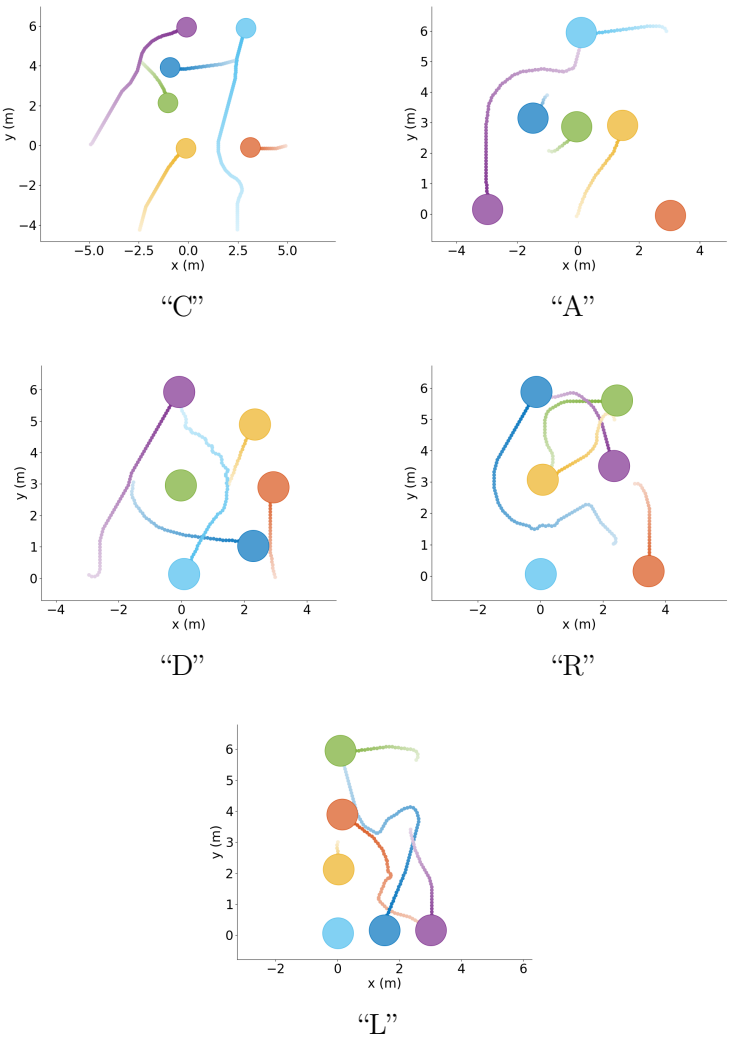


Figure 4-9: 6 agents spelling out "CADRL". Each agent is running the same GA3C-CADRL-10-LSTM policy. A centralized system randomly assigns agents to goal positions (random to ensure interaction), and each agent selects its action in a decentralized manner, using knowledge of other agents' current positions, velocities, and radii. Collision avoidance is an essential aspect of formation control.

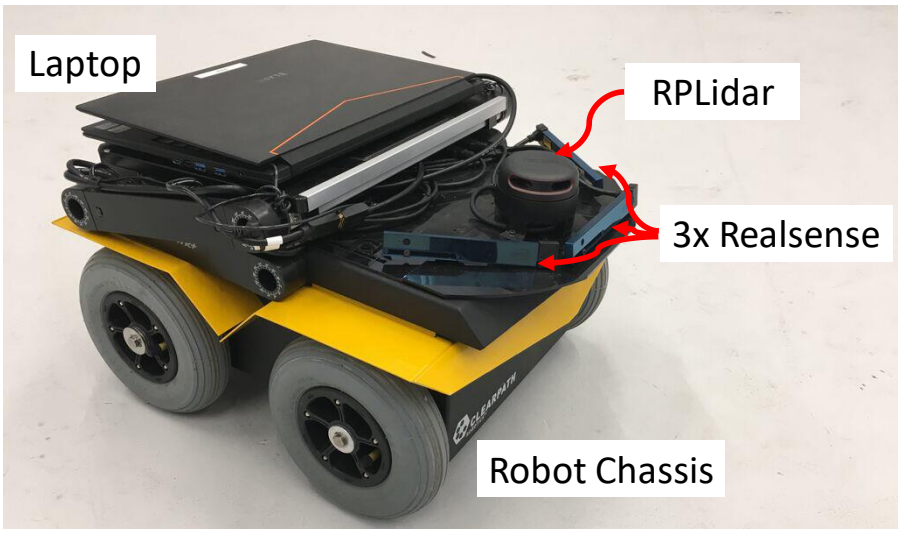


Figure 4-10: Robot hardware. The compact, low-cost ($<$ \$1000) sensing package uses a single 2D Lidar and 3 Intel Realsense R200 cameras. The total sensor and computation assembly is less than 3 inches tall, leaving room for cargo.

4.4.3 Hardware Experiments

This work implements the policy learned in simulation on two different hardware platforms to demonstrate the flexibility in the learned collision avoidance behavior and that the learned policy enables real-time decision-making. The first platform, a fleet of 4 multirotors, highlights the transfer of the learned policy to vehicles with more complicated dynamics than the unicycle kinematic model used in training. The second platform, a ground robot operating among pedestrians, highlights the policy's robustness to both imperfect perception from low-cost, on-board perception, and to heterogeneity in other agent policies, as none of the pedestrians follow one of the policies seen in training.

The hardware experiments were designed to demonstrate that the new algorithm could be deployed using realistic sensors, vehicle dynamics, and computational resources. Combined with the numerical experiments in Fig. 4-6 and Table 4.1, the hardware experiments provide evidence of an algorithm that exceeds the state of the art and can be deployed on real robots.

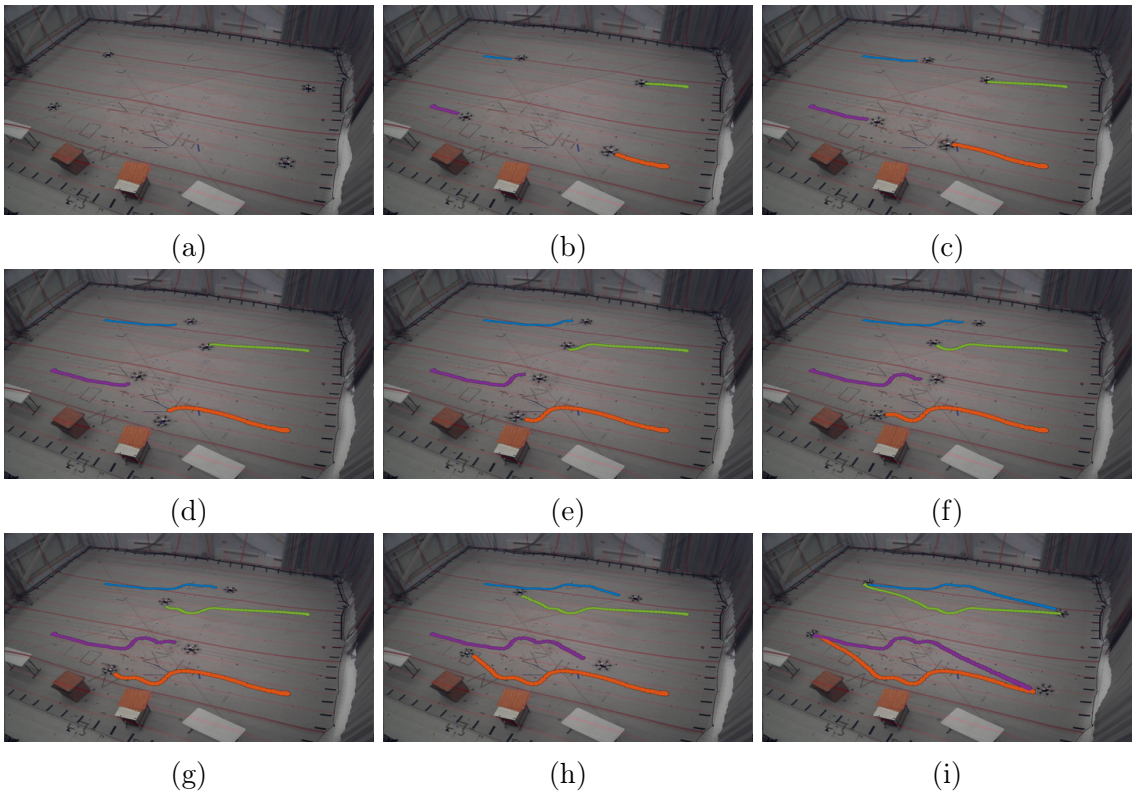
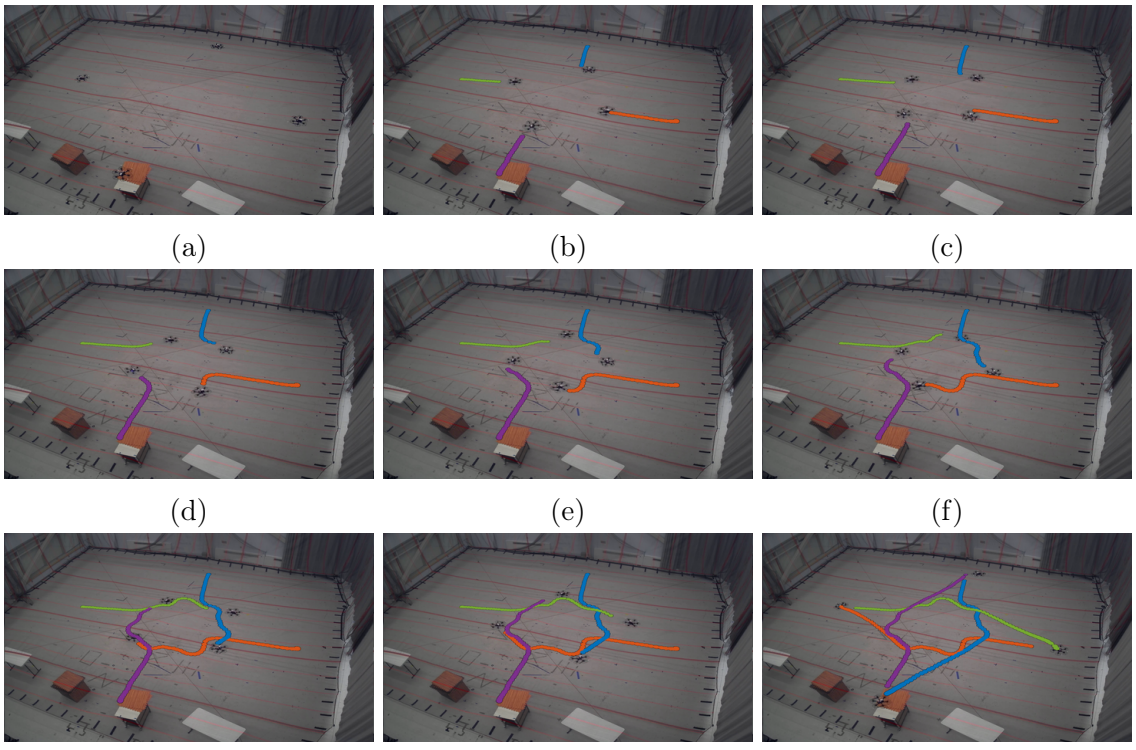


Figure 4-11: 4 Multirotors running GA3C-CADRL: 2 parallel pairs. Each vehicle's on-board controller tracks the desired speed and heading angle produced by each vehicle's GA3C-CADRL-10-LSTM policy.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 4-12: 4 Multirotors running GA3C-CADRL: 2 orthogonal pairs. The agents form a symmetric "roundabout" pattern in the center of the room, even though each vehicle has slightly different dynamics and observations of neighbors.

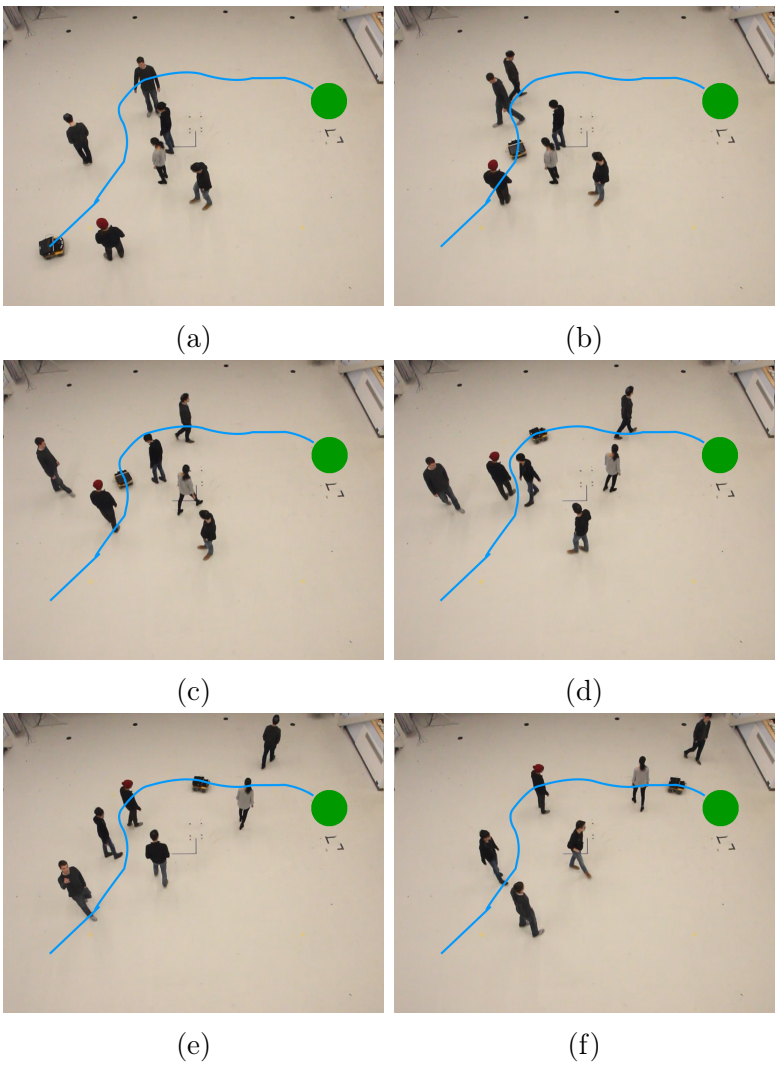


Figure 4-13: Ground robot among pedestrians. The on-board sensors are used to estimate pedestrian positions, velocities, and radii. An on-board controller tracks the GA3C-CADRL-10-LSTM policy output. The vehicle moves at human walking speed (1.2m/s), nominally.

Multiple Multirotors

A fleet of 4 multirotors with on-board velocity and position controllers resemble the agents in the simulated training environment. Each vehicle's planner receives state measurements of the other agents (positions, velocities) from a motion capture system at 200Hz [102]. At each planning step (10Hz), the planners build a state vector using other agent states, an assumed agent radius (0.5m), a preferred ego speed (0.5m/s), and knowledge of their own goal position in global coordinates. Each vehicle's planner queries the learned policy and selects the action with highest probability: a desired heading angle change and speed. A desired velocity vector with magnitude equal to the desired speed, and in the direction of the desired heading angle, is sent to the velocity controller. To smooth the near-goal behavior, the speed and heading rates decay linearly with distance to goal within 2m, and agents simply execute position control on the goal position when within 0.3m of the goal. Throughout flight, the multirotors also control to their desired heading angle; this would be necessary with a forward-facing sensor, but is somewhat extraneous given that other agents' state estimates are provided externally.

The experiments included two challenging 4-agent scenarios. In Fig. 4-11, two pairs of multirotors swap positions in parallel, much like the third column of Fig. 4-4. This policy was not trained to prefer a particular directionality – the agents demonstrate clockwise/left-handed collision avoidance behavior in the center of the room. In Fig. 4-12, the 4 vehicles swap positions passing through a common center, like the fourth column of Fig. 4-4. Unlike in simulation, where the agents' dynamics, observations, and policies (and therefore, actions) are identical, small variations in vehicle states lead to slightly different actions for each agent. However, even with these small fluctuations, the vehicles still perform "roundabout" behavior in the center.

Further examples of 2-agent swaps, and multiple repeated trials of the 4-agent scenarios are included in the videos.

Ground Robot Among Pedestrians

A GA3C-CADRL policy implemented on a ground robot demonstrates the algorithm's performance among pedestrians. We designed a compact, low-cost ($< \$1000$) sensing suite with sensors placed as to not limit the robot's cargo-carrying capability (Fig. 4-10). The sensors are a 2D Lidar (used for localization and obstacle detection), and 3 Intel Realsense R200 cameras (used for pedestrian classification and obstacle detection). Pedestrian positions and velocities are estimated by clustering the 2D Lidar's scan [103], and clusters are labeled as pedestrians using a classifier [104] applied to the cameras' RGB images [105]. A detailed description of the software architecture is in [106].

Snapshots of a particular sequence are shown in Fig. 4-13: 6 pedestrians move around between the robot's starting position and its goal (large circle) about 6m away. Between the first two frames, 3 of the pedestrians remain stationary, and the other 3 move with varying levels of cooperativeness, but these roles were not assigned and change stochastically throughout the scenario. The robot successfully navigates to its goal in the proximity of many heterogeneous agents. Other examples of safe robot navigation in challenging scenarios are available in the videos.

4.4.4 LSTM Analysis

This section provides insights into the design and inner workings of the LSTM module in Fig. 4-3 in two ways: how agent states affect the LSTM gates, and how the ordering of agents affects performance during training.

LSTM Gate Dynamics

We first analyze the LSTM of the trained GA3C-CADRL-10-LSTM network, building on [85], using notation from [94]. The LSTM weights, $\{W_i, W_f, W_o\}$, and biases, $\{b_i, b_f, b_o\}$ are updated during the training process and fixed during inference.

Recall the LSTM has three inputs: the state of agent j at time t, the previous hidden state, and the previous cell state, which are denoted $\tilde{\mathbf{s}}^o_{j,t}$, $\mathbf{h}_{t-1}$, C_{t-1}, respec-

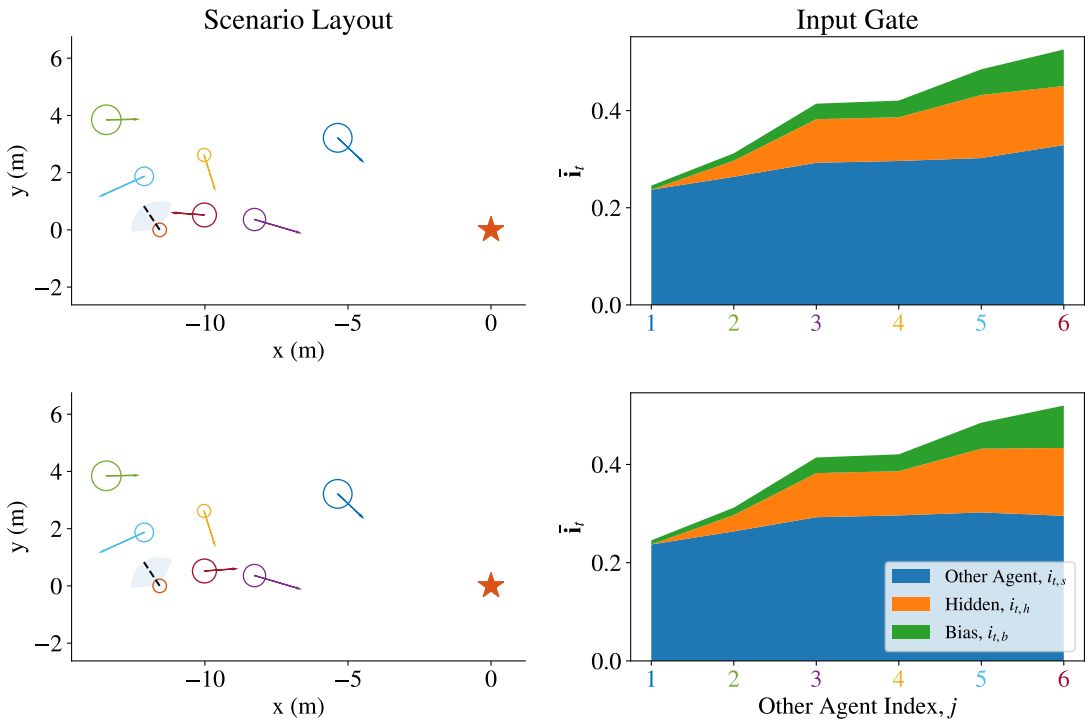


Figure 4-14: Gate Dynamics on Single Timestep. In the top row, one agent, near (-12,0), observes 6 neighboring agents. Other agent states are passed into the LSTM sequentially, starting with the furthest agent, near (-5, 3). The top right plot shows the impact of each LSTM cell input on the input gate as each agent is added to the LSTM: other agent state (bottom slice), previous hidden state (middle slice), and bias (top slice). The bottom row shows the same scenario, but the closest agent, near (-10,0), has a velocity vector away from the ego agent. Accordingly, the bottom right plot's bottom slice slightly declines from $j = 5$ to $j = 6$, but the corresponding slice increases in the top right plot. This suggests the LSTM has learned to put more emphasis on agents heading toward the ego agent, as they are more relevant for collision avoidance.

tively. Of the four gates in an LSTM, we focus on the input gate here. The input gate, $\mathbf{i}_t \in [0,1]^{n_h}$, is computed as,

$$\mathbf{i}_t = \sigma([W_{i,s},\, W_{i,h},\, W_{i,b}]^T \cdot [\tilde{\mathbf{s}}^o_{j,t},\, \mathbf{h}_t,\, b_i]), \tag{4.14}$$

where $W_i = [W_{i,h},\, W_{i,s}]$, $\;W_{i,b} = \text{diag}(b_i)$, and $n_h = 64$ is the hidden state size.

Thus, $\mathbf{i}_t = [1]^{n_h}$ when the entire *new* candidate cell state, $\tilde{C}_t$, should be used to update the cell state, C_t,

$$\tilde{C}_t = \tanh([W_{C,s}, W_{C,h}, W_{C,b}]^T \cdot [\tilde{\mathbf{s}}^o_{j,t}, \mathbf{h}_{t-1}, b_t]) \tag{4.15}$$

$$C_t = \mathbf{f}_t * C_{t-1} + \mathbf{i}_t * \tilde{C}_t, \tag{4.16}$$

where $\mathbf{f}_t$ is the value of the forget gate, computed analogously to $\mathbf{i}_t$. In other words, $\mathbf{i}_t$ with elements near 1 means that agent j is particularly important in the context of agents $[1, \ldots, j-1]$, and it will have a large impact on C_t. Contrarily, $\mathbf{i}_t$ with elements near 0 means very little of the observation about agent j will be added to the hidden state and will have little impact on the downstream decision-making.

Because $\mathbf{i}_t$ is a 64-element vector, we must make some manipulations to visualize it. First, we separate $\mathbf{i}_t$ into quantities that measure how much it is affected by each component, $\{\tilde{\mathbf{s}}^o_{j,t}, \mathbf{h}_{t-1}, b_i\}$:

$$\tilde{i}_{t,s} = ||\mathbf{i}_t - \sigma([W_{i,h}, W_{i,b}]^T \cdot [\mathbf{h}_t, b_i])||_2 \tag{4.17}$$

$$\tilde{i}_{t,h} = ||\mathbf{i}_t - \sigma([W_{i,s}, W_{i,b}]^T \cdot [\tilde{\mathbf{s}}^o_{j,t}, b_i])||_2 \tag{4.18}$$

$$\tilde{i}_{t,b} = ||\mathbf{i}_t - \sigma([W_{i,s}, W_{i,h}]^T \cdot [\tilde{\mathbf{s}}^o_{j,t}, \mathbf{h}_t])||_2 \tag{4.19}$$

$$\bar{\mathbf{i}}_t = \begin{bmatrix} i_{t,s} \\ i_{t,h} \\ i_{t,b} \end{bmatrix} = k \cdot \begin{bmatrix} \tilde{i}_{t,s} \\ \tilde{i}_{t,h} \\ \tilde{i}_{t,b} \end{bmatrix} \tag{4.20}$$

$$k = \frac{||\mathbf{i}_t||_1}{\tilde{i}_{t,s} + \tilde{i}_{t,h} + \tilde{i}_{t,b}}, \tag{4.21}$$

where the constant, k, normalizes the sum of the three components contributing to

$\bar{\mathbf{i}}_t$, and scales each by the average of all elements in $\mathbf{i}_t$.

An example scenario is shown in Fig. 4-14. A randomly generated 7-agent scenario is depicted on the left column, where the ego agent is at $(-12, 0)$, and its goal is the star at $(0, 0)$. The 6 other agents in the neighborhood are added to the LSTM in order of furthest distance to the ego agent, so the tick marks on the x-axis of the right-hand figures correspond to each neighboring agent. That is, the agent at $(-5, 3)$ is furthest from the ego agent, so it is agent $j = 0$, and the agent at $(-10, 0)$ is closest and is agent $j = 5$.

For this scenario, $\bar{\mathbf{i}}_t$ (top of the stack of three slices) starts about 0.3, and goes up and down (though trending slightly upward) as agents are added. The bottom slice corresponds to $i_{t,s}$, middle to $i_{t,h}$, and top to $i_{t,b}$.

The top and middle slices are tiny compared to the bottom slice for agent 0. This corresponds to the fact that, for $j = 0$, all information about whether that agent is relevant for decision-making is in $\tilde{\mathbf{s}}^o_{0,0}$ (bottom), since the hidden and cell states are initially blank ($h_{-1} = \mathbf{0}$). As more agents are added, the LSTM considers *both* the hidden state and current observation to decide how much of the candidate cell state to pass through – this intuition matches up with the relatively larger middle slices for subsequent agents.

The importance of the contents of $\tilde{\mathbf{s}}^o_{j,t}$ is demonstrated in the bottom row of Fig. 4-14. It considers the same scenario as the top row, but with the closest agent's velocity vector pointing away from the ego agent, instead of toward. The values of $\bar{\mathbf{i}}_t$ for all previous agents are unchanged, but the value of $\bar{\mathbf{i}}_t$ is larger when the agent is heading toward the ego agent. This is seen as an uptick between $j = 5$ and $j = 6$ in the bottom slice of the top-right figure, and a flat/slightly decreasing segment for the corresponding piece of the bottom-right figure. This observation agrees with the intuition that agents heading toward the ego agent should have a larger impact on the hidden state, and eventually on collision avoidance decision-making.

This same behavior of increased $\bar{\mathbf{i}}_t$ (specifically i_{t_s}) when the last agent was heading roughly toward the ego agent was observed in most randomly generated scenarios. Our software release will include an interactive Jupyter notebook so researchers can

analyze other scenarios of interest, or do similar analysis on their networks.

Agent Ordering Strategies

The preceding discussion on LSTM gate dynamics assumed agents are fed into the LSTM in the order of "closest last." However, there are many ways of ordering the agents.

Fig. 4-7 compares the performance throughout the training process of networks with three different ordering strategies. "Closest last" is sorted in decreasing order of agent distance to the ego agent, and "closest first" is the reverse of that. "Time to collision" is computed as the minimum time at the current velocities for two agents to collide and is often infinite when agents are not heading toward one another. The secondary ordering criterion of "closest last" was used as a tiebreaker. In all cases, $\tilde{p}_x$ (in the ego frame) was used as a third tiebreaker to preserve symmetry.

The same network architecture, differing only in the LSTM agent ordering, was trained for 1.5M episodes in 2-4 agent scenarios (Phase 1) and 0.5M more episodes in 2-10 agent scenarios (Phase 2). All three strategies yield similar performance over the first 1M training episodes. By the end of phase 1, the "closest first" strategy performs slightly worse than the other two, which are roughly overlapping.

At the change in training phase, the "closest first" performance drops substantially, and the "time to collision" curve has a small dip. This suggests that the first training phase did not produce an LSTM that efficiently combines previous agent summaries with an additional agent for these two heuristics. On the other hand, there is no noticeable dip with the "closest last" strategy. All three strategies converge to a similar final performance.

In conclusion, the choice of ordering has a second order effect on the reward curve during training, and the "closest last" strategy employed throughout this work was better than the tested alternatives. This evidence aligns with the intuition from Section 4.3.2.

4.5 Summary

This work presented a collision avoidance algorithm, GA3C-CADRL, that is trained in simulation with deep RL without requiring any knowledge of other agents' dynamics. It also proposed a strategy to enable the algorithm to select actions based on observations of a large (possibly varying) number of nearby agents, using LSTM at the network's input. The new approach was shown to outperform a classical method, another deep RL-based method, and scales better than our previous deep RL-based method as the number of agents in the environment increased. These results support the use of LSTMs to encode a varying number of agent states into a fixed-length representation of the world. Analysis of the trained LSTM provides deeper introspection into the effect of agent observations on the hidden state vector, and quantifies the effect of agent ordering heuristics on performance throughout training. The work provided an application of the algorithm for formation control, and the algorithm was implemented on two hardware platforms: a fleet of 4 fully autonomous multirotors successfully avoided collisions across multiple scenarios, and a small ground robot was shown to navigate at human walking speed among pedestrians. Combined with the numerical comparisons to prior works, the hardware experiments provide evidence of an algorithm that exceeds the state of the art and can be deployed on real robots.

A Python simulation environment can be found at `https://github.com/mit-acl/gym-collision-avoidance`, which contains several implemented policies and could be used to train new policies. The pre-trained GA3C-CADRL-10 policy wrapped as a ROS node can be found at `https://github.com/mit-acl/cadrl_ros`. The repository also includes a link to the training data set, D, needed for network initialization if training a new network. The paths were overlayed on the video/stills automatically, using `https://github.com/mfe7/video_overlay`.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Certified Adversarial Robustness for Deep Reinforcement Learning

5.1 Introduction

Deep Reinforcement Learning (RL) algorithms have achieved impressive success on robotic manipulation [107] and robot navigation in pedestrian crowds [108, 109]. Many of these systems utilize black-box predictions from Deep Neural Networks (DNN) to achieve state-of-the-art performance in prediction and planning tasks. However, the lack of formal robustness guarantees for DNNs currently limits their application in safety-critical domains, such as collision avoidance. In particular, even subtle perturbations to the input, known as *adversarial examples*, can lead to incorrect (but highly-confident) decisions by DNNs [110–112]. Furthermore, several recent works have demonstrated the danger of adversarial examples in real-world situations [113, 114], including causing an autonomous vehicle to swerve into another lane [115]. The work in this paper not only addresses deep RL algorithms' lack of robustness against network input uncertainties, but it also provides formal guarantees in the form of certificates on the solution quality.

While there are many techniques for synthesis of *empirically* robust deep RL policies [116–120], it remains difficult to synthesize a *provably* robust neural network. Instead, we leverage ideas from a related area that provides a guarantee on

how sensitive a trained network's output is to input perturbations for each nominal input [121–126]. A promising recent set of methods makes such formal robustness analysis tractable by relaxing the nonlinear constraints associated with network activations [132, 22, 127–131], unified in [133]. These relaxed methods were previously applied on computer vision/supervised learning tasks.

This work extends the tools for efficient formal neural network robustness analysis (e.g., Weng et al. [22], Singh et al. [130], Wong et al. [127]) to deep RL tasks. In RL, techniques designed for supervised learning robustness analysis would simply allow a nominal action to be *flagged* as non-robust if the minimum input perturbation exceeds a robustness threshold (e.g., the system's known level of uncertainty) – these techniques would not reason about alternative actions. Hence, we instead focus on the *robust decision-making problem*: given a known bound on the input perturbation, what is the best action to take? This aligns with the requirement that an agent *must* select an action at each step of an RL problem. Our approach uses robust optimization to consider worst-case uncertainties and provides certificates on solution quality, making it *certifiably robust*.

As a motivating example, consider the collision avoidance setting in Fig. 5-1, in which an adversary perturbs an agent's (orange) observation of an obstacle (blue). An agent following a nominal/standard deep RL policy would observe $\boldsymbol{s}_{\text{adv}}$ and select an action, a^*_{nom}, that collides with the obstacle's true position, $\boldsymbol{s}_0$, thinking that the space is unoccupied. Our proposed approach assumes a worst-case deviation of the observed input, $\boldsymbol{s}_{\text{adv}}$, bounded by $\boldsymbol{\epsilon}$, and takes the robust-optimal action, a^*_{adv}, under that perturbation, to safely avoid the true obstacle. Robustness analysis algorithms often assume $\boldsymbol{\epsilon}$ is a scalar, which makes sense for image inputs (all pixels have same scale, e.g., $0-255$ intensity). A key challenge in direct application to RL tasks is that the observation vector (network input) could have elements with substantially different scales (e.g., position, angle, joint torques) and associated measurement uncertainties, motivating our extension with vector $\boldsymbol{\epsilon}$.

This work contributes (i) the first formulation of *certifiably robust deep RL*, which uses robust optimization to consider worst-case state perturbations and provides cer-

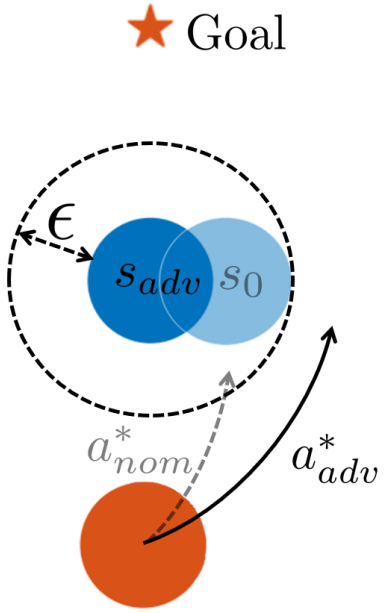


Figure 5-1: Intuition. An adversary distorts the true position, $\boldsymbol{s}_0$, of a dynamic obstacle (blue) into an adversarial observation, $\boldsymbol{s}_{\text{adv}}$. The agent (orange) only sees the adversarial input, so nominal RL policies would take a^*_{nom} to reach the goal quickly, but would then collide with the true obstacle, $\boldsymbol{s}_0$. The proposed defensive strategy considers that $\boldsymbol{s}_0$ could be anywhere inside the $\boldsymbol{\epsilon}$-ball around $\boldsymbol{s}_{\text{adv}}$, and selects the action, a^*_{adv}, with the best, worst-case outcome as calculated by a guaranteed lower bound on the value network output, which cautiously avoids the obstacle while reaching the goal. Note this is different from simply inflating the obstacle radius, since the action values contain information about environment dynamics, e.g., blue agent's cooperativeness.

tificates on solution quality, (ii) an extension of tools for efficient robustness analysis to handle variable scale inputs common in RL, and (iii) demonstrations of increased robustness to adversaries and sensor noise on cartpole and a pedestrian collision avoidance simulation.

We extend our previously published conference paper [134] with new performance guarantees (Section 5.4.4), expanded results aggregated across more scenarios (Sections 5.5.1 and 5.5.2), an extension of the algorithm into scenarios with adversarial behavior (Section 5.5.4), comparisons with a more computationally expensive method (Section 5.5.5), and visualizations that provide intuition about the robustness algorithm (Section 5.5.6).

5.2 Related work

The lack of robustness of DNNs to real-world uncertainties discovered by Szegedy et al. [110] has motivated the study of adversarial attacks (i.e., worst-case uncertainty realizations) in many learning tasks. This section summarizes adversarial attack and defense models in deep RL (see [135] for a thorough survey) and describes methods for formally quantifying DNN robustness.

5.2.1 Adversarial Attacks in Deep RL

An adversary can act against an RL agent by influencing (or exploiting a weakness in) the observation or transition models of the environment.

Observation model

Many of the techniques for attacking supervised learning networks through small image perturbations, such as those by Goodfellow et al. [136] could be used to attack image-based deep RL policies. Recent works show how to specifically craft adversarial attacks (in the input image space) against a Deep Q-Network (DQN) in RL [137, 138]. Another work applies both adversarial observation and transition perturbations [116].

Transition model

Several approaches attack an RL agent by changing parameters of the physics simulator, like friction coefficient or center of gravity, between episodes [117, 118]. Other approaches change the transition model between steps of episodes, for instance by applying disturbance forces [119, 120], or by adding a second agent that is competing against the ego agent [139]. In [140], the second agent unexpectedly learns to visually distract the ego agent rather than exerting forces and essentially becomes an observation model adversary. Thus, adversarial behavior of a second agent (the topic of *multiagent games* [141]) introduces complexities beyond the scope of this work (except a brief discussion in Section 5.5.4), which focuses on robustness to adversarial observations.

5.2.2 Empirical Defenses to Adversarial Attacks

Across deep learning in general, the goal of most existing robustness methods is to make neural networks' outputs less sensitive to small input perturbations.

Supervised Learning

First introduced for supervised learning tasks, adversarial training or retraining augments the training dataset with adversaries [142–145] to increase robustness during testing (empirically). Other works increase robustness through distilling networks [146] or comparing the output of model ensembles [147]. Rather than modifying the network training procedure, another type of approach *detects* adversarial examples through comparing the input with a binary filtered transformation of the input [148]. Although these approaches show impressive empirical success, they are often ineffective against more sophisticated adversarial attacks [149–152].

Deep RL

Many ideas from supervised learning were transferred over to deep RL to provide empirical defenses to adversaries (e.g., training in adversarial environments [116–

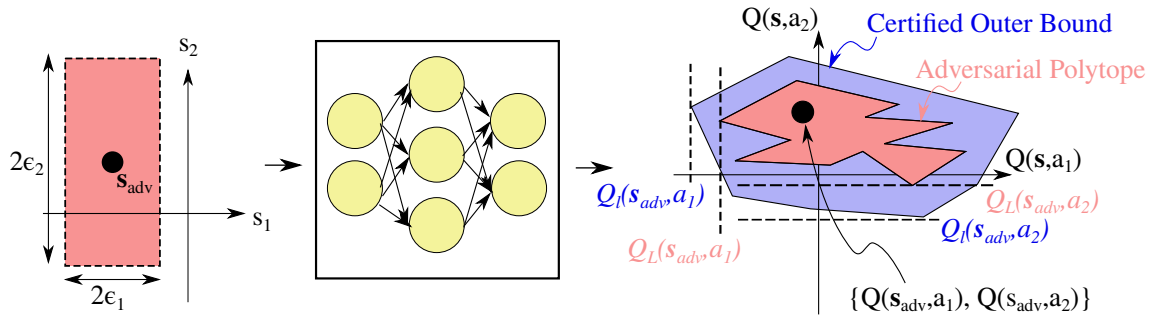


Figure 5-2: State Uncertainty Propagated Through Deep Q-Network. The red region (left) represents bounded state uncertainty (an L_∞ $\boldsymbol{\epsilon}$-ball) around the observed state, $\boldsymbol{s}_{\text{adv}}$. A neural network maps this set of possible inputs to a polytope (red) of possible outputs (Q-values in RL). This work's extension of [22] provides an outer bound (blue) on that polytope. This work then modifies the RL action-selection rule by considering lower bounds, Q_l, on the blue region for each action. In this 2-state, 2-action example, our algorithm would select action 2, since $Q_l(\boldsymbol{s}_{\text{adv}}, a_2) > Q_l(\boldsymbol{s}_{\text{adv}}, a_1)$, i.e. the worst possible outcome from action 2 is better than the worst possible outcome from action 1, given an $\boldsymbol{\epsilon}$-ball uncertainty and a pre-trained DQN.

120], using model ensembles [117, 118]). Moreover, because adversarial observation perturbations are a form of measurement uncertainty, there are close connections between Safe RL [153] and adversarial robustness. Many Safe RL (also called risk-sensitive RL) algorithms optimize for the reward under *worst-case* assumptions of environment stochasticity, rather than optimizing for the expected reward [154–156]. The resulting policies are more risk-sensitive (i.e., robust to stochastic deviations in the input space, such as sensor noise), but could still fail on algorithmically crafted adversarial examples. In other words, modifying the RL training process to directly synthesize a *provably* robust neural network remains challenging.

Instead, this work adds a defense layer on top of an already trained DQN. We provide guarantees on the robustified policy's solution quality by leveraging formal robustness analysis methods that propagate known DNN input bounds to guaranteed output bounds.

5.2.3 Formal Robustness Methods

Although synthesis of a provably robust neural network is difficult, there is a closely related body of work that can provide other formal guarantees on network robustness – namely, a guarantee on how sensitive a trained network's output is to input perturbations. The corresponding mathematical machinery to propagate an input set through a neural network allows for solving various robustness analysis problems, such as the calculation of *reachable sets*, *minimal adversarial examples*, or *robustness verification*, defined formally in Appendix A.

For example, *exact methods* can be used to find tight bounds on the maximum network output deviation, given a nominal input and bounded input perturbation. These methods rely on Satisfiability Modulo Theory (SMT) [121–123], LP/mixed-integer LP solvers [124, 125], or zonotopes [126], to propagate constraints on the input space through to the output space (exactly). The difficulty in this propagation arises through DNNs with Rectified Linear Unit (ReLU) or other nonlinear activation functions – in fact, the problem of finding the exact output bounds is NP-complete (as shown by Katz et al. [122] and Weng et al. [22]), making real-time implementations infeasible for many robotics tasks.

The formal robustness analysis approaches can be visualized in Fig. 5-2 in terms of a 2-state, 2-action deep RL problem. For a given state uncertainty set (red, left), the exact methods reason over the exact adversarial polytope (red, right), i.e., the image of the state uncertainty set through the network.

To enable scalable analysis, many researchers simultaneously proposed various relaxations of the NP-complete formulation: Salman et al. provides a unifying framework of the *convex relaxations* that we briefly summarize here [133]. A convex relaxation of the nonlinear activations enables the use of a standard LP solver to compute convex, guaranteed outer bounds, Q_{LP}, on the adversarial polytope (cf. Problem $\mathcal{C}$ in [133]). Even though this LP can be solved more efficiently than the exact problem with nonlinear activations, solving the relaxed LP is still computationally intensive.

Fortunately, the LP can be solved greedily, and thus even more efficiently, by

using just *one* linear upper bound and *one* linear lower bound for each nonlinear layer. Several seemingly different approaches (e.g., solving the dual problem [127–129], using zonotopes/polyhedra as in [130, 131], using linear outer bounds [132, 22]) were shown to lead to very similar or identical network bounds in [133]. These greedy solutions provide over-approximations (see blue region in Fig. 5-2) on the LP solution and thus, also on the adversarial polytope. One of these methods, Fast-Lin by Weng et al. [22] (which this work extends), can verify an image classification (MNIST) network's sensitivity to perturbations for a given image in $< 200ms$. As new approaches appear in the literature, it will be straightforward to "upgrade" this work's use of [22] for calculating lower bounds.

The analysis techniques described here in Section 5.2.3 are often formulated to solve the *robustness verification problem*: determine whether the calculated set of possible network outputs crosses a decision hyperplane (a line of slope 1 through the origin, in this example) – if it does cross, the classifier is deemed "not robust" to the input uncertainty. Our approach instead solves the *robust decision-making problem*: determine the best action considering worst-case outcomes, $Q_l(\boldsymbol{s}_{\text{adv}}, a_1), Q_l(\boldsymbol{s}_{\text{adv}}, a_2)$, denoted by the dashed lines in Fig. 5-2.

5.3 Background

5.3.1 Preliminaries

In RL problems[1] (see Chapter 2 for a higher-level overview), the state-action value (or, "Q-value")

$$Q(\boldsymbol{s}, a) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(t) | \mathbf{s}(t{=}0) = \boldsymbol{s}, \mathrm{a}(t{=}0) = a\right]$$

expresses the expected accumulation of future reward, r, discounted by γ, received by starting in state $\boldsymbol{s} \in \mathbb{R}^n$ and taking one of d discrete actions, $a \in \{a_0, a_1, \ldots, a_{d-1}\}$.

[1] This work considers problems with a continuous state space and discrete action space.

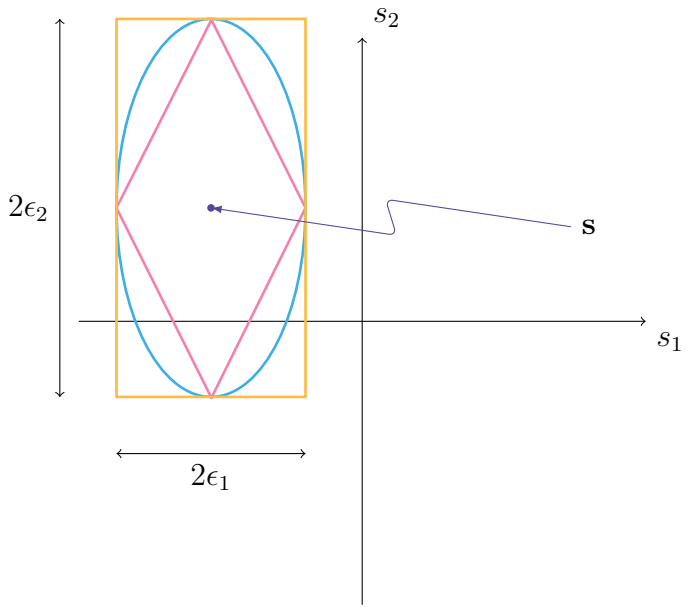


Figure 5-3: Illustration of $\boldsymbol{\epsilon}$-Ball, $\mathcal{B}_p(\boldsymbol{s}, \boldsymbol{\epsilon})$, for $n = 2$. Let $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2]$. Depending on the choice of L_p norm (e.g., L_1, L_2, and L_∞), $\mathcal{B}_p(\boldsymbol{s}, \boldsymbol{\epsilon})$ is the set of points inside the corresponding colored outline. The adversary can perturb nominal observation $\boldsymbol{s}$ to any point inside $\mathcal{B}_p(\boldsymbol{s}, \boldsymbol{\epsilon})$. The values of $\{n, \boldsymbol{\epsilon}, p\}$ are application-specific choices and the components of $\boldsymbol{\epsilon}$ need not be equal.

Let $\boldsymbol{\epsilon} \in \mathbb{R}^n_{\geq 0}$ be the maximum element-wise deviation of the state vector, and let $1 \leq p \leq \infty$ parameterize the L_p-norm. We define the set of states within this deviation as the $\boldsymbol{\epsilon}$-Ball,

$$\mathcal{B}_p(\boldsymbol{s}_0, \boldsymbol{\epsilon}) = \{\boldsymbol{s} : \lim_{\boldsymbol{\epsilon}' \to \boldsymbol{\epsilon}^+} ||(\boldsymbol{s} - \boldsymbol{s}_0) \oslash \boldsymbol{\epsilon}'||_p \leq 1\}, \tag{5.1}$$

where $\oslash$ denotes element-wise division, and the lim is only needed to handle the case where $\exists i \ \epsilon_i = 0$ (e.g., when the adversary is not allowed to perturb some component of the state, or the agent knows some component of the state vector with zero uncertainty).

An $\boldsymbol{\epsilon}$-Ball is illustrated in Fig. 5-3 for the case of $n = 2$, highlighting that different elements of the state, s_i, might have different perturbation limits, ϵ_i, and that the choice of L_p-norm affects the shape of the ball. The L_p-norm is defined as $||\boldsymbol{x}||_p = (|x_1|^p + \ldots + |x_n|^p)^{1/p}$ for $\boldsymbol{x} \in \mathbb{R}^n, 1 \leq p \leq \infty$.

5.3.2 Robustness Analysis

This work aims to find the action that maximizes state-action value under a worst-case perturbation of the observation by sensor noise or an adversary. This section explains how to efficiently obtain a lower bound on the DNN-predicted Q, given a bounded perturbation in the state space from the true state. The derivation is based on [22], re-formulated for RL.

The adversary perturbs the true state, $\boldsymbol{s}_0$, to another state, $\boldsymbol{s}_{\text{adv}} \in \mathcal{B}_{p_{\text{adv}}}(\boldsymbol{s}_0, \boldsymbol{\epsilon}_{\text{adv}})$, within the $\boldsymbol{\epsilon}_{\text{adv}}$-ball. The ego agent only observes the perturbed state, $\boldsymbol{s}_{\text{adv}}$. As displayed in Fig. 5-2, let the worst-case state-action value, Q_L, for a given action, a_j, be

$$Q_L(\boldsymbol{s}_{\text{adv}}, a_j) = \min_{\boldsymbol{s} \in \mathcal{B}_{p_{\text{adv}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{adv}})} Q(\boldsymbol{s}, a_j), \tag{5.2}$$

for all states $\boldsymbol{s}$ inside the $\boldsymbol{\epsilon}_{\text{adv}}$-Ball around the observation, $\boldsymbol{s}_{\text{adv}}$.

The goal of the analysis is to compute a guaranteed lower bound, $Q_l(\boldsymbol{s}, a_j)$, on the minimum state-action value, that is, $Q_l(\boldsymbol{s}, a_j) \leq Q_L(\boldsymbol{s}, a_j)$. The key idea is to pass interval bounds[2] $[\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)}] = [\boldsymbol{s}_{\text{adv}} - \boldsymbol{\epsilon}_{\text{adv}}, \boldsymbol{s}_{\text{adv}} + \boldsymbol{\epsilon}_{\text{adv}}]$ from the DNN's input layer to the output layer, where $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$ denote the lower and upper bounds of the pre-activation term, $\boldsymbol{z}^{(k)}$, i.e., $l_i^{(k)} \leq z_i^{(k)} \leq u_i^{(k)} \forall i \in 1, ..., u_k$, in the k-th layer with u_k units of an m-layer DNN. When passing interval bounds through a ReLU activation[3], $\sigma(\cdot)$, the upper and lower pre-ReLU bounds of each element could either both be positive $(l_r^{(k)}, u_r^{(k)} > 0)$, negative $(l_r^{(k)}, u_r^{(k)} < 0)$, or positive and negative $(l_r^{(k)} < 0, u_r^{(k)} > 0)$, in which the ReLU status is called *active, inactive* or *undecided*, respectively. In the active and inactive case, bounds are passed directly to the next layer. In the

[2]Element-wise $\pm\boldsymbol{\epsilon}_{\text{adv}}$ can cause overly conservative categorization of ReLUs for $p < \infty$. p is accounted for later in the Algorithm in Eq. (5.12).

[3]Although this work considers DNNs with ReLU activations, the formulation could be extended to general activation functions via more recent algorithms [132].

undecided case, the output of the ReLU is bounded linearly above and below:

$$\sigma(z_r^{(k)})_{|l_r^{(k)},u_r^{(k)}} = \begin{cases} [z_r^{(k)}, z_r^{(k)}] \\ \qquad \text{if } l_r^{(k)}, u_r^{(k)} > 0, \text{“active”} \\ [0,0] \\ \qquad \text{if } l_r^{(k)}, u_r^{(k)} < 0, \text{“inactive”} \\ [\frac{u_r^{(k)}}{u_r^{(k)}-l_r^{(k)}} z_r^{(k)}, \frac{u_r^{(k)}}{u_r^{(k)}-l_r^{(k)}}(z_r^{(k)} - l_r^{(k)})] \\ \qquad \text{if } l_r^{(k)}<0, u_r^{(k)}>0, \text{“undecided”}, \end{cases} \tag{5.3}$$

for each element, indexed by r, in the k-th layer.

The identity matrix, D, is introduced as the ReLU status matrix, H as the lower/upper bounding factor, W as the weight matrix, b as the bias in layer (k) with r or j as indices, and the pre-ReLU-activation, $z_r^{(k)}$, is replaced with $W_{r,:}^{(k)}\boldsymbol{s} + b_r^{(k)}$. The ReLU bounding is then rewritten as

$$\begin{aligned} & D_{r,r}^{(k)}(W_{r,j}^{(k)} s_j + b_r^{(k)}) \\ \leq \; & \sigma(W_{r,j}^{(k)} s_j + b_r^{(k)}) \\ \leq \; & D_{r,r}^{(k)}(W_{r,j}^{(k)} s_j + b_r^{(k)} - H_{r,j}^{(k)}), \end{aligned} \tag{5.4}$$

where

$$D_{r,r}^{(k)} = \begin{cases} 1 & \text{if } l_r^{(k)}, u_r^{(k)}>0; \\ 0 & \text{if } l_r^{(k)}, u_r^{(k)}<0, \\ \frac{u_r^{(k)}}{u_r^{(k)}-l_r^{(k)}} & \text{if } l_r^{(k)}<0, u_r^{(k)}>0; \end{cases} \tag{5.5}$$

$$H_{r,j}^{(k)} = \begin{cases} l_r^{(k)} & \text{if } l_r^{(k)}<0, u_r^{(k)}>0, A_{j,r}^{(k)}<0; \\ 0 & \text{otherwise.} \end{cases} \tag{5.6}$$

Using these ReLU relaxations, a guaranteed lower bound of the state-action value for

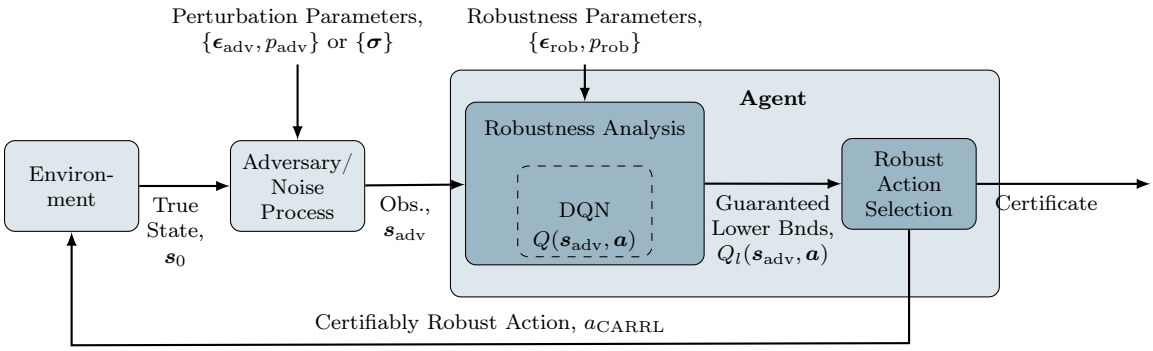


Figure 5-4: System Architecture. During online execution, an agent observes a state, $\boldsymbol{s}_{\text{adv}}$, corrupted by an adversary or noise process (constrained to $\boldsymbol{s}_{\text{adv}} \in \mathcal{B}_{p_{\text{adv}}}(\boldsymbol{s}_0, \boldsymbol{\epsilon}_{\text{adv}})$. A trained Deep RL algorithm, e.g., DQN [157], predicts the state-action values, Q. The robustness analysis module computes a lower bound of the network's predicted state-action values of each discrete action: Q_l, w.r.t. a robustness threshold $\boldsymbol{\epsilon}_{\text{rob}}$ and $L_{p_{\text{rob}}}$-norm in the input space. The agent takes the action, a_{CARRL}, that maximizes the lower bound, i.e., is best under a worst-case deviation in the input space, and returns a *certificate* bounding a_{CARRL}'s sub-optimality.

a single state $\boldsymbol{s} \in \mathcal{B}_p(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon})$ (based on [22]) is:

$$\bar{Q}_l(\boldsymbol{s}, a_j) = A^{(0)}_{j,:}\boldsymbol{s} + b^{(m)}_j + \sum_{k=1}^{m-1} A^{(k)}_{j,:}(\boldsymbol{b}^{(k)} - H^{(k)}_{:,j}), \tag{5.7}$$

where the matrix A contains the network weights and ReLU activation, recursively for all layers: $A^{(k-1)} = A^{(k)}W^{(k)}D^{(k-1)}$, with identity in the final layer: $A^{(m)} = \mathbb{1}$.

Unlike the exact DNN output, the bound on the relaxed DNN's output in Eq. (5.7) can be minimized across an $\boldsymbol{\epsilon}$-ball in closed form (as described in Section 5.4.3), which is a key piece of this work's real-time, robust decision-making framework.

5.4 Approach

This work develops an add-on, certifiably robust defense for existing Deep RL algorithms to ensure robustness against sensor noise or adversarial examples during test time.

5.4.1 System architecture

In an offline training phase, an agent uses a deep RL algorithm, here DQN [157], to train a DNN that maps non-corrupted state observations, $\boldsymbol{s}_0$, to state-action values, $Q(\boldsymbol{s}_0, a)$. Action selection during training uses the nominal cost function, $a^*_{\text{nom}} = \text{argmax}_{a_j} Q(\boldsymbol{s}_0, a_j)$.

Figure 5-4 depicts the system architecture of a standard model-free RL framework with the added-on robustness module. During online execution, the agent only receives corrupted state observations from the environment. The robustness analysis node uses the DNN architecture, DNN weights, W, and robustness hyperparameters, $\boldsymbol{\epsilon}_{\text{rob}}, p_{\text{rob}}$, to compute lower bounds on possible Q-values for robust action selection.

5.4.2 Optimal cost function under worst-case perturbation

We assume that the training process causes the network to converge to the optimal value function, $Q^*(\boldsymbol{s}_0, a)$ and focus on the challenge of handling perturbed observations during execution. Thus, we consider robustness to an adversary that perturbs the true state, $\boldsymbol{s}_0$, within a small perturbation, $\boldsymbol{\epsilon}_{\text{adv}}$, into the worst-possible state observation, $\boldsymbol{s}_{\text{adv}}$. The adversary assumes that the RL agent follows a nominal policy (as in, e.g., DQN) of selecting the action with highest Q-value at the current observation. A worst possible state observation, $\boldsymbol{s}_{\text{adv}}$, is therefore any one which causes the RL agent to take the action with lowest Q-value in the true state, $\boldsymbol{s}_0$:

$$\begin{aligned}\boldsymbol{s}_{\text{adv}} \in \{\boldsymbol{s} : \boldsymbol{s} \in \mathcal{B}_{p_{\text{adv}}}(\boldsymbol{s}_0, \boldsymbol{\epsilon}_{\text{adv}}) \text{ and } \\ \operatorname*{argmax}_{a_j} Q(\boldsymbol{s}, a_j) = \operatorname*{argmin}_{a_j} Q(\boldsymbol{s}_0, a_j)\}.\end{aligned} \tag{5.8}$$

This set could be computationally intensive to compute and/or empty – an approximation is described in Section 5.4.6.

After the agent receives the state observation picked by the adversary, the agent selects an action. Instead of trusting the observation (and thus choosing the worst action for the true state), the agent could leverage the fact that the true state, $\boldsymbol{s}_0$,

must be somewhere inside an $\boldsymbol{\epsilon}_{\text{adv}}$-Ball around $\boldsymbol{s}_{\text{adv}}$ (i.e., $\boldsymbol{s}_0 \in \mathcal{B}_{p_{\text{adv}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{adv}})$).

However, in this work, the agent assumes $\boldsymbol{s}_0 \in \mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}}))$, where we make a distinction between the adversary and robust agent's parameters. This distinction introduces hyperparameters $\boldsymbol{\epsilon}_{\text{rob}}, p_{\text{rob}}$ that provide further flexibility in the defense algorithm's conservatism, that do not necessarily have to match what the adversary applies. Nonetheless, for the sake of providing guarantees, the rest of Section 5.4 assumes $\boldsymbol{\epsilon}_{\text{rob}} = \boldsymbol{\epsilon}_{\text{adv}}$ and $p_{\text{rob}} = p_{\text{adv}}$ to ensure $\mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})) = \mathcal{B}_{p_{\text{adv}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{adv}}))$. Empirical effects of tuning $\boldsymbol{\epsilon}_{\text{rob}}$ to other values are explored in Section 5.5.

The agent evaluates each action by calculating the worst-case Q-value under all possible true states. In accordance with the robust decision making problem, the robust-optimal action, a^*, is defined here as one with the highest Q-value under the worst-case perturbation,

$$a^* = \underset{a_j}{\text{argmax}} \underbrace{\min_{\boldsymbol{s} \in \mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})} Q(\boldsymbol{s}, a_j)}_{Q_L(\boldsymbol{s}_{\text{adv}}, a_j)} . \tag{5.9}$$

As described in Section 5.2, computing $Q_L(\boldsymbol{s}_{\text{adv}}, a_j)$ exactly is too computationally intensive for real-time decision-making.

Thus, this work proposes the algorithm Certified Adversarial Robustness for Deep Reinforcement Learning (CARRL). In CARRL, the action, a_{CARRL}, is selected by approximating $Q_L(\boldsymbol{s}_{\text{adv}}, a_j)$ with $Q_l(\boldsymbol{s}_{\text{adv}}, a_j)$, its guaranteed lower bound across all possible states $\boldsymbol{s} \in \mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})$, so that:

$$a_{\text{CARRL}} = \underset{a_j}{\text{argmax}}\, Q_l(\boldsymbol{s}_{\text{adv}}, a_j), \tag{5.10}$$

where Eq. (5.16) below defines $Q_l(\boldsymbol{s}_{\text{adv}}, a_j)$ in closed form. Conditions for optimality ($a^* = a_{\text{CARRL}}$) are described in Section 5.4.4.

5.4.3 Robustness analysis with vector-ϵ-ball perturbations

To solve Eq. (5.10) when $Q(\boldsymbol{s}, a_j)$ is represented by a DNN, we adapt the formulation from [22]. Most works in adversarial examples, including [22], focus on perturbations on image inputs, in which all channels have the same scale (e.g., grayscale images with pixel intensities in $[0, 255]$). More generally, however, input channels could be on different scales (e.g., joint torques, velocities, positions). Existing robustness analysis methods require choosing a scalar ϵ_{rob} that bounds the uncertainty across all input channels; in general, this could lead to unnecessarily conservative behavior, as some network inputs might be known with zero uncertainty, or differences in units could make uncertainties across channels incomparable. Hence, this work computes bounds on the network output under perturbation bounds specific to each network input channel, as described in a vector $\boldsymbol{\epsilon}_{\text{rob}}$ with the same dimension as $\boldsymbol{s}$.

To do so, we minimize $\bar{Q}_l(\boldsymbol{s}, a_j)$ across *all* states in $\mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})$, where $\bar{Q}_l(\boldsymbol{s}, a_j)$ was defined in Eq. (5.7) as the lower bound on the Q-value for a *particular* state $\boldsymbol{s} \in \mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})$. This derivation uses a vector $\boldsymbol{\epsilon}_{\text{rob}}$ (instead of scalar ϵ_{rob} as in [22]):

$$
\begin{aligned}
Q_l(\boldsymbol{s}_{\text{adv}}, a_j) &= \min_{\boldsymbol{s} \in \mathcal{B}_p(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})} \left(\bar{Q}_l(\boldsymbol{s}, a_j) \right) && (5.11) \\
&= \min_{\boldsymbol{s} \in \mathcal{B}_p(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})} \left(A^{(0)}_{j,:} \boldsymbol{s} + \underbrace{b^{(m)}_j + \sum_{k=1}^{m-1} A^{(k)}_{j,:} (\boldsymbol{b}^{(k)} - H^{(k)}_{:,j})}_{=:\Gamma} \right) && (5.12) \\
&= \min_{\boldsymbol{s} \in \mathcal{B}_p(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})} \left(A^{(0)}_{j,:} \boldsymbol{s} \right) + \Gamma && (5.13) \\
&= \min_{\boldsymbol{y} \in \mathcal{B}_p(\boldsymbol{0}, \boldsymbol{1})} \left(A^{(0)}_{j,:} (\boldsymbol{y} \odot \boldsymbol{\epsilon}_{\text{rob}}) \right) + A^{(0)}_{j,:} \boldsymbol{s}_{\text{adv}} + \Gamma && (5.14) \\
&= \min_{\boldsymbol{y} \in \mathcal{B}_p(\boldsymbol{0}, \boldsymbol{1})} \left((\boldsymbol{\epsilon}_{\text{rob}} \odot A^{(0)}_{j,:}) \boldsymbol{y} \right) + A^{(0)}_{j,:} \boldsymbol{s}_{\text{adv}} + \Gamma && (5.15) \\
&= -||\boldsymbol{\epsilon}_{\text{rob}} \odot A^{(0)}_{j,:}||_q + A^{(0)}_{j,:} \boldsymbol{s}_{\text{adv}} + \Gamma, && (5.16)
\end{aligned}
$$

with $\odot$ denoting element-wise multiplication. From Eq. (5.11) to Eq. (5.12), we substitute in Eq. (5.7). From Eq. (5.12) to Eq. (5.13), we introduce the placeholder variable Γ that does not depend on $\boldsymbol{s}$. From Eq. (5.13) to Eq. (5.14), we substi-

tute $\boldsymbol{s} := \boldsymbol{y} \odot \boldsymbol{\epsilon}_{\text{rob}} + \boldsymbol{s}_{\text{adv}}$, to shift and re-scale the observation to within the unit ball around zero, $\boldsymbol{y} \in \mathcal{B}_p(\mathbf{0}, \mathbf{1})$. The maximization in Eq. (5.15) is equivalent to a L_q-norm in Eq. (5.16) by the definition of the dual norm $||\boldsymbol{z}||_q = \{\sup \boldsymbol{z}^T \boldsymbol{y} : ||\boldsymbol{y}||_p \leq 1\}$ and the fact that the L_q norm is the dual of the L_p norm for $p, q \in [1, \infty)$ (with $1/p + 1/q = 1$). Equation (5.16) is inserted into Eq. (5.10) to calculate the CARRL action in closed form.

Recall from Section 5.2 that the bound Q_l is the greedy (one linear upper and lower bound per activation) solution of an LP that describes a DNN with ReLU activations. In this work, we refer to the full (non-greedy) solution to the primal convex relaxed LP as Q_{LP} (cf. Problem $\mathcal{C}$, **LP-All** in [133]).

To summarize, the relationship between each of the Q-value terms is:

$$Q(\boldsymbol{s}_{\text{adv}}, a_j) \geq Q_L(\boldsymbol{s}_{\text{adv}}, a_j) \geq Q_{LP}(\boldsymbol{s}_{\text{adv}}, a_j) \geq Q_l(\boldsymbol{s}_{\text{adv}}, a_j) \tag{5.17}$$

$$Q(\boldsymbol{s}_{\text{adv}}, a_j) \geq \bar{Q}_l(\boldsymbol{s}_{\text{adv}}, a_j) \geq Q_l(\boldsymbol{s}_{\text{adv}}, a_j). \tag{5.18}$$

5.4.4 Guarantees on Action Selection

Avoiding a Bad Action

Unlike the nominal DQN rule, which could be tricked to take an arbitrarily bad action, the robust-optimal decision-rule in Eq. (5.9) can avoid bad actions provided there is a better alternative in the $\boldsymbol{\epsilon}_{\text{rob}}$-Ball.

Claim 1: If for some q', $\exists \boldsymbol{s}' \in \mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}}), a'$ s.t. $Q(\boldsymbol{s}', a') \leq q'$ and $\exists a''$ s.t. $\forall \boldsymbol{s} \in \mathcal{B}_{p_{\text{rob}}}(\boldsymbol{s}_{\text{adv}}, \boldsymbol{\epsilon}_{\text{rob}})\ Q(\boldsymbol{s}, a'') > q'$, then $a^* \neq a'$.

In other words, if action a' is sufficiently bad for some nearby state, $\boldsymbol{s}'$, and at least one other action a'' is better for all nearby states, the robust-optimal decision rule will not select the bad action a' (but DQN might). This is because $Q_L(\boldsymbol{s}_{\text{adv}}, a') \leq q'$, $Q_L(\boldsymbol{s}_{\text{adv}}, a'') > q' \Rightarrow \text{argmax}_{a_j} Q_L(\boldsymbol{s}_{\text{adv}}, a_j) \neq a'$.

Matching the Robust-Optimal Action

While Claim 1 refers to the robust-optimal action, the action returned by CARRL is the same as the robust-optimal action (returned by a system that can compute the exact lower bounds) under certain conditions,

$$\underbrace{\underset{a_j}{\operatorname{argmax}}\, Q_l(\boldsymbol{s}_{\text{adv}}, a_j)}_{a_{\text{CARRL}}} \stackrel{?}{=} \underbrace{\underset{a_j}{\operatorname{argmax}}\, Q_L(\boldsymbol{s}_{\text{adv}}, a_j)}_{a^*}. \tag{5.19}$$

Claim 2: CARRL selects the robust-optimal action if the robustness analysis process satisfies $Q_l = g(Q_L)$, where g is a strictly monotonic function, where Q_l, Q_L are written without their arguments, $\boldsymbol{s}_{\text{adv}}, a_j$. A special case of this conditions is when the analysis returns a tight bound, i.e., $Q_l(\boldsymbol{s}_{\text{adv}}, a_j) = Q_L(\boldsymbol{s}_{\text{adv}}, a_j)$. Using the Fast-Lin-based approach in this work, for a particular observation, confirmation that all of the ReLUs are "active" or "inactive" in Eq. (5.3) would provide a tightness guarantee.

In cases where Claim 2 is not fulfilled, but Claim 1 is fulfilled, CARRL is not guaranteed to select the robust-optimal action, but will still reason about all possible outcomes, and empirically selects a better action than a nominal policy across many settings explored in Section 5.5.

Note that when $\boldsymbol{\epsilon}_{\text{rob}} = \mathbf{0}$, no robustness is applied, so both the CARRL and robust-optimal decisions reduce to the DQN decision, since $Q_l(\boldsymbol{s}_{\text{adv}}, a_j) = Q_L(\boldsymbol{s}_{\text{adv}}, a_j) = Q(\boldsymbol{s}_{\text{adv}}, a_j)$.

Claim 3: CARRL provides a *certificate* of the chosen action's sub-optimality,

$$0 \leq Q(\mathbf{s}_0,\, a^*) - Q(\mathbf{s}_0,\, a_{\text{CARRL}}) \leq \underbrace{\left(\max_a Q_u(\mathbf{s}_{\text{adv}},\, a)\right) - Q_l(\mathbf{s}_{\text{adv}},\, a_{\text{CARRL}})}_{\text{Sub-optimality Certificate}}, \tag{5.20}$$

where Q_u is an upper bound on Q computed analogously to Q_l.

This bound captures the uncertainty in the true state (considering the whole ϵ-ball through Q_u, Q_l), and the unknown action a^* that is optimal for the true state.

5.4.5 Probabilistic Robustness

The discussion so far considered cases where the state perturbation is known to be bounded (as in, e.g., many adversarial observation perturbation definitions [136], stochastic processes with finite support. However, in many other cases, the observation uncertainty is best modeled by a distribution with infinite support (e.g., Gaussian). To be fully robust to this class of uncertainty (including very low probability events) CARRL requires setting $\epsilon_{rob,i} = \infty$ for the unbounded state elements, $\boldsymbol{s}_i$.

For instance, for a Gaussian sensor model with known standard deviation of measurement error, $\boldsymbol{\sigma}_{sensor}$, one could set $\boldsymbol{\epsilon}_{\text{rob}} = 2\boldsymbol{\sigma}_{sensor}$ to yield actions that account for the worst-case outcome with 95% confidence. In robotics applications, for example, $\boldsymbol{\sigma}_{sensor}$ is routinely provided in a sensor datasheet. Implementing this type of probabilistic robustness only requires a sensor model for the observation vector and a desired confidence bound to compute the corresponding $\boldsymbol{\epsilon}_{\text{rob}}$.

5.4.6 Adversaries

To evaluate the learned policy's robustness to deviations of the input in an $\boldsymbol{\epsilon}_{\text{adv}}$-Ball, we pass the true state, $\boldsymbol{s}_0$, through an adversarial/noise process to compute $\boldsymbol{s}_{\text{adv}}$, as seen in Fig. 5-4. Computing an adversarial state $\boldsymbol{s}_{\text{adv}}$ exactly, using Eq. (5.8), is computationally intensive. Instead, we use a Fast Gradient Sign method with Targeting (FGST) [113] to approximate the adversary from Eq. (5.8). FGST chooses a state $\hat{\boldsymbol{s}}_{\text{adv}}$ on the L_∞ $\boldsymbol{\epsilon}_{\text{adv}}$-Ball's perimeter to encourage the agent to take the nominally worst action, $\text{argmin}_{a_j} Q(\boldsymbol{s}_0, a_j)$. Specifically, $\hat{\boldsymbol{s}}_{\text{adv}}$ is picked along the direction of sign of the lowest cross-entropy loss, $\mathcal{L}$, between $\boldsymbol{y}_{\text{adv}}$, a one-hot encoding of the worst action at $\boldsymbol{s}_0$, and $\boldsymbol{y}_{\text{nom}}$, the softmax of all actions' Q-values at $\boldsymbol{s}_0$. The loss is taken w.r.t. the proposed observation $\boldsymbol{s}$:

$$\boldsymbol{y}_{\text{adv}} = [\mathbb{1}\{a_i = \text{argmin}_{a_j} Q(\boldsymbol{s}_0, a_j)\}] \in \mathbb{U}^d \tag{5.21}$$

$$Q_{\text{nom}} = [Q(\boldsymbol{s}_0, a_j) \forall j \in \{0, \ldots, d-1\}] \in \mathbb{R}^d \tag{5.22}$$

$$\boldsymbol{y}_{\text{nom}} = \text{softmax}(Q_{\text{nom}}) \in \Delta^d \tag{5.23}$$

$$\hat{\boldsymbol{s}}_{\text{adv}} = \boldsymbol{s}_0 - \boldsymbol{\epsilon}_{\text{adv}} \odot \text{sign}(\nabla_{\boldsymbol{s}} \mathcal{L}(\boldsymbol{y}_{\text{adv}}, \boldsymbol{y}_{\text{nom}})), \tag{5.24}$$

where $\mathbb{U}^d$ denotes a one-hot vector of dimension d, and Δ^d denotes the standard d-simplex.

In addition to adversarial perturbations, Section 5.5 also considers uniform noise perturbations, where $\boldsymbol{s}_{\text{adv}} \sim \texttt{Unif}\left([\boldsymbol{s}_0 - \boldsymbol{\sigma}, \boldsymbol{s}_0 + \boldsymbol{\sigma}]\right)$.

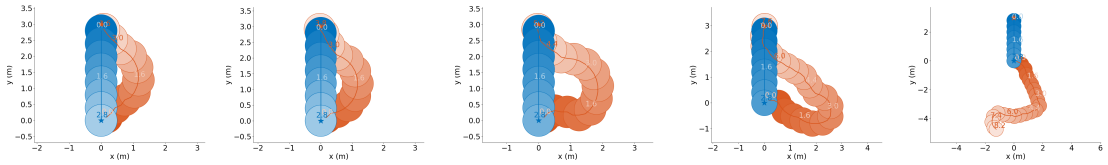
5.4.7 Certified vs. Verified Terminology

Now that we have introduced our algorithm, we can more carefully describe what we mean by a *certifiably robust* algorithm.

First, we clarify what it means to be *verified*. In the context of neural network robustness, a network is *verified robust* for some nominal input if no perturbation within some known set can change the network's decision from its nominal decision. A *complete* verification algorithm would correctly label the network as either *verified robust* or *verified non-robust* for any nominal input and perturbation set. In practice, verification algorithms often involve network relaxations, and the resulting algorithms are *sound* (any time the network gives an answer, the answer is true), but sometimes return that they can not verify either property.

A *certificate* is some piece of information that allows an algorithm to quantify its solution quality. For example, Boyd et al. use a dual feasible point as a *certificate* to bound the primal solution, which allows one to compute a bound on any feasible point's suboptimality even when the optimal value of the primal problem is unknown [158]. When an algorithm provides a certificate, the literature commonly refers to the algorithm as *certified*, *certifiable*, or *certifiably* _____ (if it provides a *certificate of* _____*-ness*).

In this work, we use a robust optimization formulation to consider worst-case possibilities on state uncertainty (Eqs. (5.9) and (5.10)); this makes our algorithm *robust*. Furthermore, we provide a certificate on how sub-optimal the robust action recommended by our algorithm is, with respect to the optimal action at the (unknown) true state (Eq. (5.20)). Thus, we describe our algorithm as *certifiably robust* or that



(a) $\epsilon_{\text{rob}} = 0.0$ (b) $\epsilon_{\text{rob}} = 0.1$ (c) $\epsilon_{\text{rob}} = 0.46$ (d) $\epsilon_{\text{rob}} = 1.0$ (e) $\epsilon_{\text{rob}} = 2.0$

Figure 5-5: Increase of conservatism with ϵ_{rob}. An agent (orange) following the CARRL policy avoids a dynamic, non-cooperative obstacle (blue) that is observed without noise. An increasing robustness parameter ϵ_{rob} (left to right) increases the agent's conservatism, i.e., the agent avoids the obstacle with a greater safety distance.

it provides *certified adversarial robustness*.

As an aside, some works in computer vision/robot perception propose *certifiably correct* algorithms [159, 160]. In those settings, there is often a combinatorial problem that *could* be solved optimally with enough computation time, but practitioners prefer an algorithm that efficiently computes a (possibly sub-optimal) solution and comes with a certificate of solution quality. Our setting differs; the state uncertainty means that recovering the optimal action for the true state is not a matter of computation time. Thus, we do not aim for *correctness* (choosing the optimal action for the true state), but rather choose an action that maximizes worst-case performance.

5.5 Experimental Results

The key result is that while imperfect observations reduce the performance of a nominal deep RL algorithm, our proposed algorithm, CARRL, recovers much of the performance by adding robustness during execution. Robustness against perturbations from an adversary or noise process during execution is evaluated in two simulated domains: collision avoidance among dynamic, decision-making obstacles [161] and cartpole [162]. This section also describes the impact of the $\boldsymbol{\epsilon}_{\text{rob}}$ hyperparameter, the ability to handle behavioral adversaries, a comparison with another analysis technique, and provides intuition on the tightness of the lower bounds.

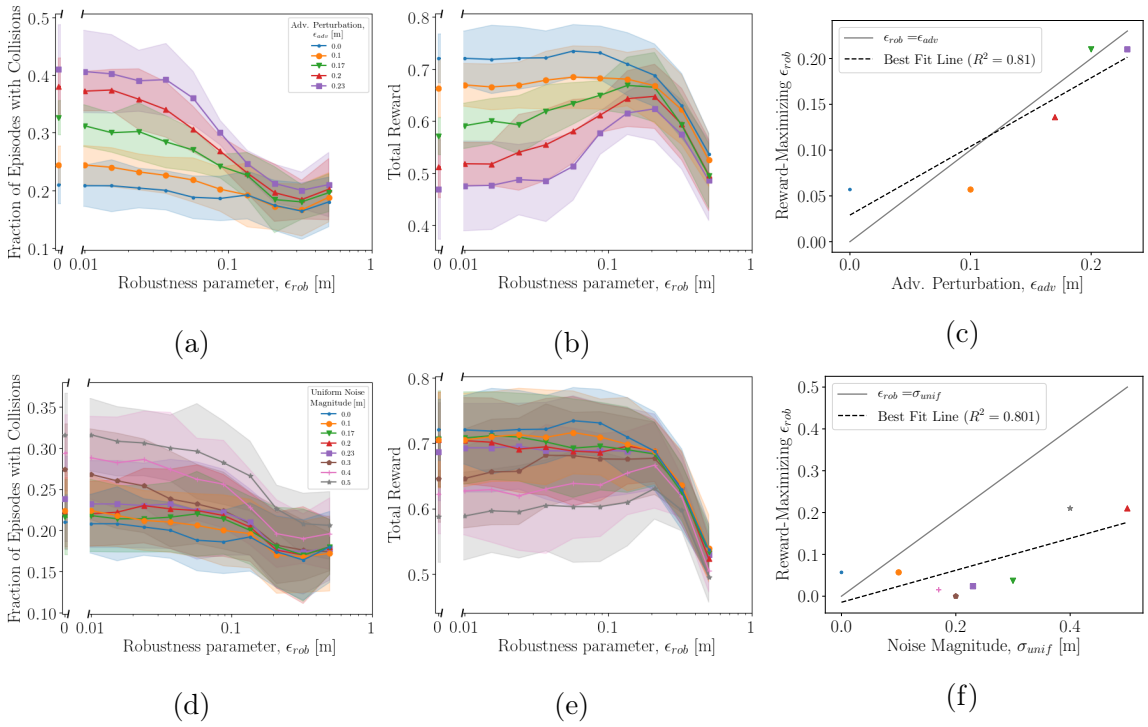


Figure 5-6: Robustness against adversarial attacks (top row) and noise (bottom row). Increasing the robustness parameter, ϵ_{rob}, decreases the number of collisions in the presence of adversarial attacks (a), or noise (d) for various perturbation magnitudes, $\epsilon_{\text{adv}}, \sigma_{\text{unif}}$. CARRL also recovers substantial amounts of reward lost due to imperfect observations as ϵ_{rob} increases (b,e). The choice of ϵ_{rob} to maximize reward in the presence of a particular adversary or noise process can be guided by the models in (c,f).

5.5.1 Collision Avoidance Domain

Among the many RL tasks, a particularly challenging safety-critical task is collision avoidance for a robotic vehicle among pedestrians. Because learning a policy in the real world is dangerous and time consuming, this work uses a *gym*-based [162] kinematic simulation environment [161] for learning pedestrian avoidance policies. In this work, the RL policy controls one of two agents with 11 discrete actions: change of heading angle evenly spaced between $[-\pi/6, +\pi/6]$ and constant velocity $v = 1$ m/s. The environment executes the selected action under unicycle kinematics, and controls the other agent from a diverse set of fixed policies (static, non-cooperative, ORCA [163], GA3C-CADRL [109]). The sparse reward is 1 for reaching the goal, -0.25 for colliding (and 0 otherwise, i.e., zero reward is received in cases where the agent does not reach its goal in a reasonable amount of time). The observation vector includes the CARRL agent's goal, each agent's radius, and the other agent's x-y position and velocity, with more detail in [109]. In this domain, robustness and perturbations are applied only on the measurement of the other agent's x-y position (i.e., $\boldsymbol{\epsilon}_{\text{rob}} = \epsilon_{\text{rob}} \cdot [0 \ \ldots \ 0 \ 1 \ 1 \ 0 \ \ldots 0]$) – an example of CARRL's ability to handle uncertainties of varying scales.

A non-dueling DQN policy was trained with 2, 64-unit layers with the following hyperparameters: learning rate $2.05\texttt{e}{-4}$, ϵ-greedy exploration ratio linearly decaying from 0.5 to 0.05, buffer size $152\texttt{e}3$, $4\texttt{e}5$ training steps, and target network update frequency, $10\texttt{e}3$. The hyperparameters were found by running 100 iterations of Bayesian optimization with Gaussian Processes [164] on the maximization of the sparse training reward.

After training, CARRL is added onto the trained DQN policy. Intuition on the resulting CARRL policy is demonstrated in Fig. 5-5. The figure shows the trajectories of the CARRL agent (orange) for increasing ϵ_{rob} values in a scenario with unperturbed observations. With increasing ϵ_{rob} (toward right), the CARRL agent accounts for increasingly large worst-case perturbations of the other agent's position. Accordingly, the agent avoids the dynamic agent (blue) increasingly conservatively, i.e., selects

actions that leave a larger safety distance. When ϵ_{rob}=2.0, the CARRL agent is overly conservative – it "runs away" and does not reach its goal, which is explained more in Section 5.5.6.

Figure 5-6 shows that the nominal DQN policy is not robust to the perturbation of inputs. In particular, increasing the magnitude of adversarial, or noisy perturbation, $\epsilon_{\text{adv}}, \sigma_{\text{unif}}$, drastically 1) increases the average number of collisions (as seen in Figs. 5-6a and 5-6d, respectively, at $\epsilon_{\text{rob}} = 0$) and 2) decreases the average reward (as seen in Figs. 5-6b and 5-6e). The results in Fig. 5-6 are evaluated in scenarios where the other agent is non-cooperative (i.e., travels straight to its goal position at constant velocity) and each datapoint represents the average of 100 trials across 5 random seeds that determine the agents' initial/goal positions and radii (shading represents ± 1 standard deviation of average value per seed).

Next, we demonstrate that CARRL recovers performance. Increasing the robustness parameter ϵ_{rob} decreases the number of collisions under varying magnitudes of noise, or adversarial attack, as seen in Figs. 5-6a and 5-6d. Because collisions affect the reward function, the received reward also increases with an increasing robustness parameter $\epsilon_{\text{rob}} < \sim 0.1$ under varying magnitudes of perturbations. As expected, the effect of the proposed defense is highest under large perturbations, as seen in the slopes of the curves ϵ_{adv}=0.23 (violet) and σ_{unif}=0.5 (gray).

Since the CARRL agent selects actions more conservatively than a nominal DQN agent, it is able to successfully reach its goal instead of colliding like a nominal DQN agent does under many scenarios with noisy or adversarial perturbations. However, the effect of overly conservative behavior seen in Figs. 5-5d and 5-5e appears in Fig. 5-6 for $\epsilon_{\text{rob}} > \sim 0.2$, as the reward drops significantly. This excessive conservatism for large $\boldsymbol{\epsilon}_{\text{rob}}$ can be partially explained by the fact that highly conservative actions may move the agent's position observation into states that are far from what the network was trained on, which breaks CARRL's assumption of a perfectly learned Q-function. Further discussion about the conservatism inherent in the lower bounds is discussed in Section 5.5.6.

Figures 5-6c and 5-6f illustrate further intuition on choosing ϵ_{rob}. Figure 5-6c

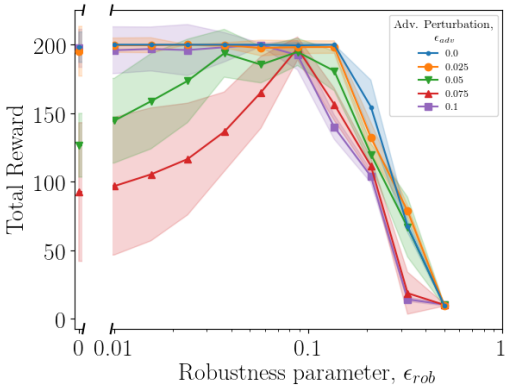
demonstrates a strong correlation between the attack magnitude ϵ_{adv} and the best (i.e., reward-maximizing) robustness hyperparameter ϵ_{rob} under that attack magnitude from Fig. 5-6b. In the case of uniform noise, the correlation between ϵ_{rob} and σ_{unif} is weaker, because the FGST adversary chooses an input state on the perimeter of the $\boldsymbol{\epsilon}_{\text{adv}}$-Ball, whereas uniform noise samples lie inside the $\boldsymbol{\sigma}_{unif}$-Ball.

The flexibility in setting ϵ_{rob} enables CARRL to capture uncertainties of various magnitudes in the input space, e.g., ϵ_{rob} could be adapted on-line to account for a perturbation magnitude that is unknown a priori, or to handle time-varying sensor noise.

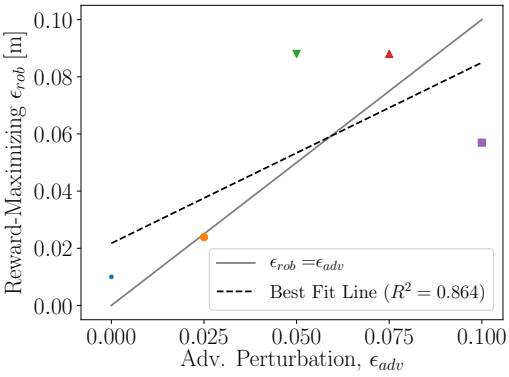
5.5.2 Cartpole Domain

In the cartpole task [162, 165], the reward is the number of time steps (capped at 200) that a pole remains balanced upright ($\pm 12^\circ$ from vertical) on a cart that can translate along a horizontal track. The state vector, $\boldsymbol{s} = \left[p_{cart}, v_{cart}, \theta_{pole}, \dot{\theta}_{pole}\right]^T$ and action space, $\boldsymbol{a} \in \{\text{push cart left}, \text{push cart right}\}$ are defined in [162]. A 2-layer, 4-unit network was trained in an environment without any observation perturbations using an open-source DQN implementation [166] with Bayesian Optimization used to find training hyperparameters. The trained DQN is evaluated against perturbations of various magnitudes, shown in Fig. 5-7. In this domain, robustness and perturbations are applied to all states equally, i.e., $\boldsymbol{\epsilon}_{\text{adv}} = \epsilon_{\text{adv}} \cdot \mathbb{1} \in \mathbb{R}^4$, $\boldsymbol{\epsilon}_{\text{rob}} = \epsilon_{\text{rob}} \cdot \mathbb{1} \in \mathbb{R}^4$ and $\boldsymbol{\sigma}_{unif} = \sigma_{unif} \cdot \mathbb{1} \in \mathbb{R}^4$.

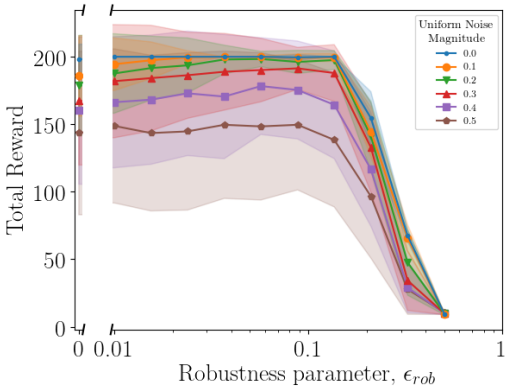
Each curve in Figs. 5-7a and 5-7c corresponds to the average reward received under different magnitudes of adversarial, or uniform noise perturbations, respectively. The reward of a nominal DQN agent drops from 200 under perfect measurements to 105 under adversarial perturbation of magnitude $\epsilon_{\text{adv}} = 0.075$ (blue and red reward at x-axis, $\epsilon_{\text{rob}} = 0$ in Fig. 5-7a) or to 145 under uniform noise of magnitude $\sigma_{\text{unif}} = 0.5$ (Fig. 5-7c). For $\epsilon_{\text{rob}} > 0$ (moving right along x-axis), the CARRL algorithm considers an increasingly large range of states in its worst-case outcome calculation. Accordingly, the algorithm is able to recover some of the performance lost due to imperfect observations. For example, with $\epsilon_{\text{adv}} = 0.075$ (red triangles), CARRL achieves 200



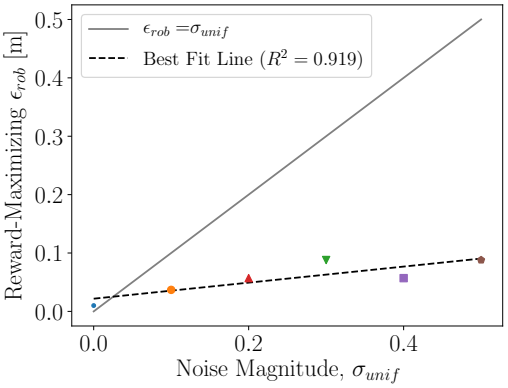
(a) Robustness to Adversaries (Cartpole)



(b) Choosing ϵ_{rob} in Presence of Adversary (Cartpole)



(c) Robustness to Sensor Noise (Cartpole)



(d) Choosing ϵ_{rob} in Presence of Noise (Cartpole)

Figure 5-7: Results on Cartpole. CARRL recovers performance (measured by reward received) by adding robustness under various magnitudes of adversarial (a) and uniform noise (c) perturbations. Each curve in (a,c) correspond to a different magnitude of perturbation, and $\epsilon_{\text{rob}} = 0$ corresponds to zero robustness, i.e., DQN. For all adversary/noise magnitudes, CARRL becomes overly conservative for large ϵ_{rob}, and the performance degrades. Thus, choosing the best ϵ_{rob} for a particular perturbation magnitude can be guided by the curves in (b,d).

reward using CARRL with $\epsilon_{\text{rob}} = 0.1$. The ability to recover performance is due to CARRL selecting actions that consider the worst-case state (e.g., a state in which the pole is closest to falling), rather than fully trusting the perturbed observations.

However, there is again tradeoff between robustness and conservatism. For large values of ϵ_{rob}, the average reward declines steeply for all magnitudes of adversaries and noise, because CARRL considers an excessively large set of worst-case states (more detail provided in Section 5.5.6).

The reward-maximizing choice of ϵ_{rob} for a particular class and magnitude of perturbations is explored in Figs. 5-7b and 5-7d. Similarly to the collision avoidance domain, the best choice of ϵ_{rob} is close to ϵ_{adv} under adversarial perturbations and less than σ_{unif} under uniform noise perturbations.

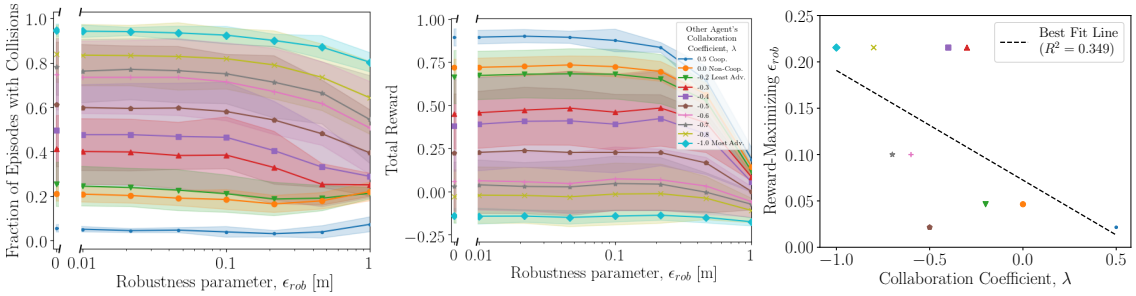
These results use 200 random seeds causing different initial conditions.

5.5.3 Computational Efficiency

For the cartpole task, one forward pass with bound calculation takes on average 0.68 ± 0.06ms, which compares to a forward pass of the same DQN (nominal, without calculating bounds) of 0.24 ± 0.03ms; for collision avoidance, it takes 1.85 ± 1.62ms (CARRL) and 0.30 ± 0.04ms (DQN), all on one i7-6700K CPU. In our implementation, the bound of all actions (i.e., 2 or 11 discrete actions for the cartpole and collision avoidance domain, respectively) are computed in parallel. While the DQNs used in this work are relatively small, [22] shows that the runtime of Fast-Lin scales linearly with the network size, and a recent GPU implementation offers faster performance [167], suggesting CARRL could be implemented in real-time for higher-dimensional RL tasks, as well.

5.5.4 Robustness to Behavioral Adversaries

In addition to observational perturbations, many real-world domains also require interaction with other agents, whose *behavior* could be adversarial. In the collision avoidance domain, this can be modeled by an environment agent who actively tries



(a) (b) (c)

Figure 5-8: Robustness to Adversarial Behavior. Each curve shows a different magnitude of adversarial *behavior* of another agent in the collision avoidance task. The adversarially behaving other agents (negative collaboration coefficient) are able to cause many collisions with a CARRL agent trained among cooperative and non-cooperative agents. Although CARRL was not explicitly designed to handle behavioral adversaries, CARRL can reduce the number of collisions by providing robustness in the other agent's position measurement. CARRL's effect on reward (b) is not as strong as in Fig. 5-6, but the reward-maximizing $\epsilon_{\text{rob}} > 0$ against all adversaries (c), and there is a trend of larger ϵ_{rob} working well against stronger adversaries.

to collide with the CARRL agent, as opposed to the various cooperative or neutral behavior models seen in the training environment described earlier. Although the CARRL formulation does not explicitly consider behavioral adversaries, one can introduce robustness to this class of adversarial perturbation through robustness in the observation space, namely by specifying uncertainty in the other agent's position. In other words, requiring the CARRL agent to consider worst-case positions of another agent while selecting actions causes the CARRL agent to maintain a larger spacing, which in turn prevents the adversarially behaving agent from getting close enough to cause collisions.

In the collision avoidance domain, we parameterize the adversarial "strength" by a collaboration coefficient, λ, where $\lambda = 0.5$ corresponds to a nominal ORCA agent (that does half the collision avoidance), $\lambda = 0$ corresponds to a non-cooperative agent that goes straight toward its goal, and $\lambda \in [-1, 0)$ corresponds to an adversarially behaving agent. Adversarially behaving agents sample from a Bernoulli distribution (every 1 second) with parameter $|\lambda|$. If the outcome is 1, the adversarial agent chooses actions directly aiming into the CARRL agent's projected future position, otherwise,

it chooses actions moving straight toward its goal position. Thus, $\lambda = -1$ means the adversarial agent is always trying to collide with the CARRL agent.

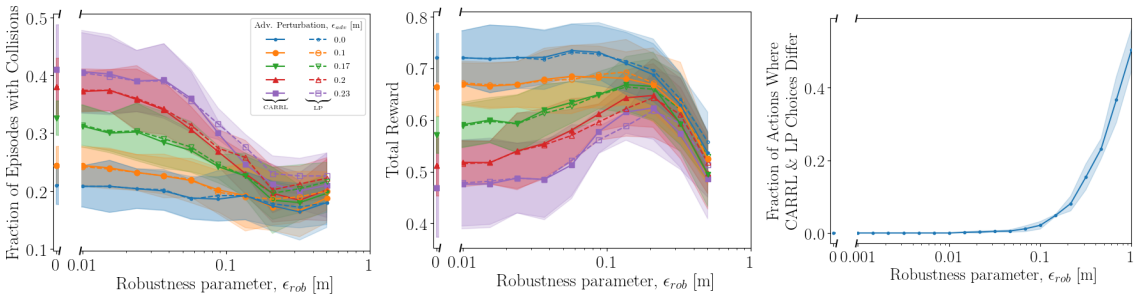
The idea of using observational robustness to protect against behavioral uncertainty is quantified in Fig. 5-8, where each curve corresponds to a different behavioral adversary. Increasing the magnitude of $\lambda < 0$ (increasingly strong adversaries) causes collisions with increasing frequency, since the environment transition model is increasingly different from what was seen during training. Increasing CARRL's robustness parameter ϵ_{rob} leads to a reduction in the number of collisions, as seen in Fig. 5-8a. Accordingly, the reward received increases for certain values of $\epsilon_{\text{rob}} > 0$ (seen strongest in red, violet curves; note y-axis scale is wider than in Fig. 5-6b). Although the impact on the reward function is not as large as in the observational uncertainty case, Fig. 5-8c shows that the reward-maximizing choice of ϵ_{rob} has some negative correlation the adversary's strength (i.e., the defense should get stronger against a stronger adversary).

The same trade-off of over-conservatism for large ϵ_{rob} exists in the behavioral adversary setting. Because there are perfect observations in this experiment (just like training), CARRL has minimal effect when the other agent is cooperative (blue, $\lambda = 0.5$). The non-cooperative curve (orange, $\lambda = 0$) matches what was seen in Fig. 5-6 with $\epsilon_{\text{adv}} = 0$ or $\sigma_{\text{unif}} = 0$. 100 test cases with 5 seeds were used.

5.5.5 Comparison to LP Bounds

As described in Section 5.2, the convex relaxation approaches (e.g., Fast-Lin) provide relatively loose, but fast-to-compute bounds on DNN outputs. Equation (5.17) relates various bound tightnesses theoretically, which raises the question: how much better would the performance of an RL agent be, given more computation time to better approximate the worst-case outcome, Q_L?

In Fig. 5-9, we compare the performance of an agent following the CARRL decision rule, versus one that approximates $Q_L(\boldsymbol{s}_{\text{adv}}, \boldsymbol{a})$ with the full (non-greedy) primal convex relaxed LP (in the collision avoidance domain with observational perturbations). Our implementation is based on the **LP-ALL** implementation provided in [132]. The



(a) (b) (c)

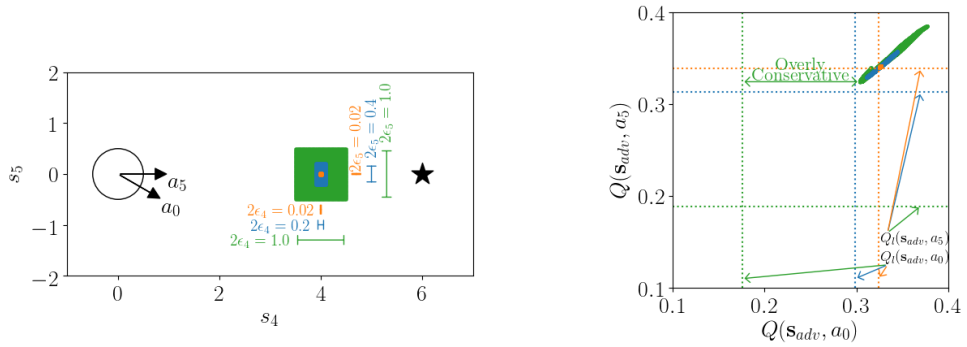
Figure 5-9: Greedy Convex Bounds vs. LP. Using efficient but greedy bounds on the adversarial polytope, CARRL's action-selection produces similar performance to that of action-selection using the less relaxed LP (cf. **LP-ALL** in [132]) (solid vs. dashed curves in each color). In (c), CARRL and LP select identical actions on $> 99\%$ of timesteps (across 60 episodes) for small ϵ_{rob}, but the two methods diverge as the greedy bounds become overly conservative. Thus, for small-to-moderate ϵ_{rob}, the extra computation required for the LP does not have substantial effect on decision-making.

key takeaway is that there is very little difference: the CARRL curves are solid and the LP curves are dashed, for various settings of adversarial perturbation, ϵ_{adv}. The small difference in the two algorithms is explained by the fact that CARRL provides extra conservatism versus the LP (CARRL accounts for a worse worst-case than the LP), so the CARRL algorithm performs slightly better when ϵ_{rob} is set too small, and slightly worse when ϵ_{rob} is too large for the particular observational adversary.

This result is further explained by Fig. 5-9c, where it is shown that CARRL and LP-based decision rules choose the same action $> 99\%$ of the time for $\epsilon < 0.1$, meaning in this experiment, the extra time spent computing the tighter bounds has little impact on the RL agent's decisions.

5.5.6 Intuition on Bounds

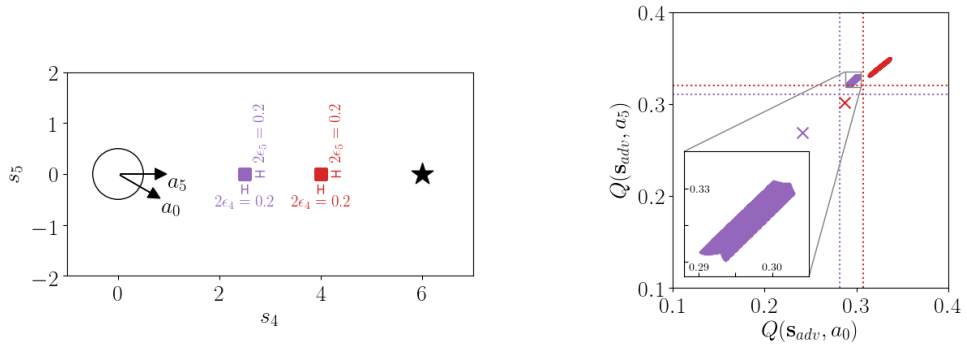
Visual inspection of actual adversarial polytopes and the corresponding efficiently computed bounds provides additional intuition into the CARRL algorithm. The state uncertainty's mapping into an adversarial polytope in the Q-value space is visualized in Figs. 5-10 and 5-11. In Fig. 5-10a, a CARRL agent observes another agent (of the same size) positioned somewhere in the concentric, colored regions. The state



(a) Position Uncertainty of Another Agent [m]

(b) Corresponding Bounds & Sampled DQN Outputs

Figure 5-10: Influence of $\boldsymbol{\epsilon}_{\text{rob}}$ on Q-Values. In (a), the CARRL agent (◯) has a goal (★) at (6,0) and decides between two actions, a_0 and a_5. A second agent could be centered somewhere in the colored regions, corresponding to different values of $\boldsymbol{\epsilon}_{\text{rob}}$. In (b) are the corresponding Q-values for those possible states (orange, blue green regions in top-right), exhaustively sampled in each $\boldsymbol{\epsilon}_{\text{rob}}$-Ball. As $\boldsymbol{\epsilon}_{\text{rob}}$ increases, the spread of possible Q-values increases. CARRL's lower bounds on each action, $Q_l(\boldsymbol{s}_{\text{adv}}, a_0)$, $Q_l(\boldsymbol{s}_{\text{adv}}, a_5)$, are depicted by the dotted lines. Conservatism is measured by the gap between the dashed line and the left/bottom-most sampled point. For small $\boldsymbol{\epsilon}_{\text{rob}}$ (orange), the bounds are tight; for moderate $\boldsymbol{\epsilon}_{\text{rob}}$ (blue) are moderately conservative, and for large $\boldsymbol{\epsilon}_{\text{rob}}$ (green), the linear approximation of ReLU degrades, causing excessive conservatism.



(a) Position Uncertainty of Another Agent [m]

(b) Corresponding Bounds & Sampled DQN Outputs

Figure 5-11: Influence of $\boldsymbol{s}_{\text{adv}}$ on Q-Values. For the same $\boldsymbol{\epsilon}_{\text{rob}}$, the spread of Q-values are shown for two examples of $\boldsymbol{s}_{\text{adv}}$. When the other agent is close (purple), the Q-values are lower than when the other agent is far (red). A closer look at one of the non-convex adversarial polytopes is inset in (b). Moreover, if one instead used the heuristic of simply inflating the other agent's radius, the Q-values would lie at the ×'s – in both cases, radius inflation is more conservative (further toward bottom-left) than CARRL.

uncertainty, drawn for various values of $\boldsymbol{\epsilon}_{\text{rob}}$ with L_∞ norm manifests itself in Fig. 5-10b as a region of possible $Q(\boldsymbol{s}_{\text{adv}}, a_j)$ values, for each action, a_j. Because there are only two dimensions of state uncertainty, we can exhaustively sample Q-values for these states. To visualize the corresponding Q-value region in 2D, consider just two actions, a_0, a_5 (right-most and straight actions, respectively).

The efficiently computed lower bounds, Q_l for each action are shown as dotted lines for each of the $\boldsymbol{\epsilon}_{\text{rob}}$-Balls. Note that for small $\boldsymbol{\epsilon}_{\text{rob}}$ (orange) the bounds are quite tight to the region of possible Q-values. A larger $\boldsymbol{\epsilon}_{\text{rob}}$ region (blue) leads to looser (but still useful) bounds, and this case also demonstrates $\boldsymbol{\epsilon}_{\text{rob}}$ with non-uniform components ($\epsilon_4 = 0.1, \epsilon_5 = 0.2$, where $(4, 5)$ are the indices of the other agent's position in the state vector) For large $\boldsymbol{\epsilon}_{\text{rob}}$ (green), the bounds are very loose, as the lowest sampled Q-value for a_0 is 0.3, but $Q_l(\boldsymbol{s}_{\text{adv}}, a_0) = 0.16$.

This increase in conservatism with $\boldsymbol{\epsilon}_{\text{rob}}$ is explained by the formulation of Fast-Lin, in that linear bounds around an "undecided" ReLU become looser for larger input uncertainties. That is, as defined in Eq. (5.3), the lower linear bound of an undecided ReLU in layer k, neuron i is $\frac{u_i^{(k)} \cdot l_i^{(k)}}{u_i^{(k)} + l_i^{(k)}} < 0$ for $z_i^{(k)} = l_i^{(k)}$, whereas a ReLU can never output a negative number. This under-approximation gets more extreme for large $\epsilon_{rob,i}$, since $u_i^{(k)}, l_i^{(k)}$ become further apart in the input layer, and this conservatism is propagated through the rest of the network. Per the discussion in [132], using the same slope for the upper and lower bounds has computational advantages but can lead to conservatism. The CROWN [132] algorithm generalizes this notion and allows adaptation in the slopes; connecting CROWN with CARRL is a minor change left for future work.

Moreover, expanding the possible uncertainty magnitudes without excessive conservatism and while maintaining computational efficiency could be an area of future research. Nonetheless, the lower bounds in CARRL were sufficiently tight to be beneficial across many scenarios.

In addition to $\boldsymbol{\epsilon}_{\text{rob}}$, Fig. 5-11a considers the impact of variations in $\boldsymbol{s}_{\text{adv}}$: when the other agent is closer to the CARRL agent (purple), the corresponding Q-values in Fig. 5-11b are lower for each action, than when the other agent is further away

(red). Note that the shape of the adversarial polytope (region of Q-value) can be quite complicated due to the high dimensionality of the DNN (inset of Fig. 5-11b). Furthermore, the $\times$ symbols correspond to the Q-value if the agent simply inflated the other agent's radius to account for the whole region of uncertainty. This heuristic is highly conservative (and domain-specific), as the $\times$'s have lower Q-values than the efficiently computed lower bounds for each action in these examples.

5.6 Summary

This work adapted deep RL algorithms for application in safety-critical domains, by proposing an add-on *certifiably robust* defense to address existing failures under adversarially perturbed observations and sensor noise. The proposed extension of robustness analysis tools from the verification literature into a deep RL formulation enabled efficient calculation of a lower bound on Q-values, given the observation perturbation/uncertainty. These guaranteed lower bounds were used to efficiently solve a robust optimization formulation for action selection to provide maximum performance under worst-case observation perturbations. Moreover, the resulting policy comes with a *certificate* of solution quality, even though the true state and optimal action are unknown to the certifier due to the perturbations. The resulting policy (added onto trained DQN networks) was shown to improve robustness to adversaries and sensor noise, causing fewer collisions in a collision avoidance domain and higher reward in cartpole. Furthermore, the proposed algorithm was demonstrated in the presence of adversaries in the behavior space, compared against a more time-intensive alternative, and visualized for particular scenarios to provide intuition on the algorithm's conservatism.

Chapter 6

Conclusion

6.1 Summary of Contributions

This thesis provides algorithms that enable robust autonomous robot navigation in human environments by investigating the following questions: (i) How to guide motion plans with scene context when the goal is only described semantically, and there is no prior map? (ii) How to navigate safely among pedestrians, of which there may be a large, possibly time-varying number? (iii) How to use learned policies under adversarial observation uncertainty?

Chapter 3 presented an algorithm for learning to utilize context in a structured environment in order to inform an exploration-based planner. The new approach, called Deep Cost-to-Go (DC2G), represents scene context in a semantic gridmap, learns to estimate which areas are beneficial to explore to quickly reach the goal, and then plans toward promising regions in the map. The efficient training algorithm requires zero training in simulation: a context extraction model is trained on a static dataset, and the creation of the dataset is highly automated. The algorithm outperforms pure frontier exploration by 189% across 42 real test house layouts. A high-fidelity simulation shows that the algorithm can be applied on a robot with a forward-facing camera to successfully complete last-mile deliveries.

Chapter 4 presented a collision avoidance algorithm, GA3C-CADRL, that is trained in simulation with deep RL without requiring any knowledge of other agents' dynam-

ics. It also proposed a strategy to enable the algorithm to select actions based on observations of a large (possibly varying) number of nearby agents, using LSTM at the network's input. The new approach was shown to outperform a classical method, another deep RL-based method, and scales better than our previous deep RL-based method as the number of agents in the environment increased.

Chapter 5 adapted deep RL algorithms for application in safety-critical domains, by proposing an add-on *certifiably robust* defense to address existing failures under adversarially perturbed observations and sensor noise. The proposed extension of robustness analysis tools from the verification literature into a deep RL formulation enabled efficient calculation of a lower bound on Q-values, given the observation perturbation/uncertainty. These guaranteed lower bounds were used to efficiently solve a robust optimization formulation for action selection to provide maximum performance under worst-case observation perturbations. Moreover, the resulting policy comes with a *certificate* of solution quality, even though the true state and optimal action are unknown to the certifier due to the perturbations. The resulting policy (added onto trained DQN networks) was shown to improve robustness to adversaries and sensor noise, causing fewer collisions in a collision avoidance domain and higher reward in cartpole. Furthermore, the proposed algorithm was demonstrated in the presence of adversaries in the behavior space, compared against a more time-intensive alternative, and visualized for particular scenarios to provide intuition on the algorithm's conservatism.

Overall, these contributions address key challenges in robust robot navigation, but there are still several remaining issues before we see robots everywhere. Toward that goal, possible directions of future work are described below.

6.2 Future Directions

6.2.1 Multi-Modal Cost-to-Go Estimation

Chapter 3 learned to estimate the MAP cost-to-go, by training a DNN with pixel-wise L_1 loss compared to particular ground truth maps. However, the planning algorithm could benefit from the full distribution of the cost-to-go conditioned on the current region of the map observed so far. In other words, rather than following the DNN's single output image, it would be beneficial to identify multiple promising routes toward the goal. Possible formulations could use a CVAE [168], Pixel-RNN [169], or multi-modal GAN [170]. This direction would enhance the level of scene understanding by robots and could reduce the amount of exploration needed before finding the goal.

6.2.2 Hybrid Planning for Multiagent Collision Avoidance

The deep RL formulation of multiagent collision avoidance used in Chapter 4 lacks formal guarantees on avoiding collisions or deadlock, which can be an issue for safety-critical systems. Moreover, the one-step policy lookup is not constrained to respect vehicle dynamics, which can cause an overly confident control sequence. MPC-based approaches [171], on the other hand, can be formulated to return a dynamically feasible trajectory, but typically have limited ability to reason about pedestrian interaction online, due to computational overhead. Thus, a planning framework which leverages the benefits of both MPC and RL would provide a way to respect vehicle dynamics (and possibly guaranteed collision constraints), while leveraging some of the computation-saving information learned through the offline RL process. Initial work on hybrid planners in this domain switch between model-based (PID, Social-Force-based) and RL planners given the world configuration [172, 173].

6.2.3 Online Estimation and Adaptation for Certified RL Robustness

The choice of hyperparameter ϵ_{rob} in Chapter 5 has a substantial impact on system performance. When ϵ_{rob} is too high, the algorithm is overly conservative, and when ϵ_{rob} is too low, the algorithm does not properly account for system uncertainty. While exhaustive offline search could work in some problems, there is a desire to estimate what ϵ_{rob} should be set to automatically. This could be done in a model-free (adapting the parameter until the rewards reach a maximum) or model-based (modeling the sources of uncertainty and setting the robustness accordingly) fashion.

6.2.4 Long-Term Objectives

As we look further ahead to enable robust robot navigation among humans, the topics investigated in this thesis could be expanded to more broadly consider:

Collective Knowledge Absorption

The most fascinating property about navigation through crowds is that the harder the problem becomes, the more examples of solutions there are. This goes against almost any other robotics problem – normally as the task gets harder, there are fewer experts to show or explain a solution. We should create algorithms that make the most of this property, by absorbing the collective knowledge of the crowd. Rather than searching for a navigation algorithm that will handle every scenario, we should give robots the ability to *absorb knowledge* that is leaked by nearby humans. For instance, when a robot reaches a new hallway, no single pedestrian will solve the exact same navigation problem as the robot. But, the crowd could provide dozens of examples in the exact test environment that *suggest how* to solve the navigation problem. Developing a framework to quickly piece together the clues humans are using will allow robots to navigate the same environments humans so effortlessly flow through.

Don't Avoid the Crowd; Join It

We often frame navigation as a reach-avoid problem: reach the goal while avoiding the crowd. However, the key to avoiding individual things in the crowd is to *join the crowd*; moving with the flow of traffic can shield the robot from oncoming pedestrians and obstacles. Given this idea, we could re-frame the reach-avoid problem: consider the set to reach as all the places in the crowd's flow that *transport* the robot toward its goal. This raises many interesting questions: how to join and leave a moving crowd smoothly and safely? Once in the flow, how can the robot move around effectively? More fundamentally, how can a robot infer what behaviors make up a crowd? The objective could be to create a robot that could navigate through MIT's infinite corridor at 10:05am on a Monday, where a stopped robot would clog traffic and a moving robot would need to avoid walls and pedestrians. To be fully robust in densely packed human environments, robots will need to understand the relationships between crowd density and navigation behavior, and to leverage the *benefits* offered by the crowd.

Robustness *from* Learning

A common viewpoint is that learning-based methods *can* give great performance, at the cost of robustness. We should instead think of robustness – the ability to handle situations that could not be modeled perfectly – as a property that *comes from* learning. For example, we could create incredibly robust motion planners, if only robots had human-level perception abilities. Learning may eventually enable human-level perception, which would mean planners gained *robustness from learning.*

Nonetheless, this thesis recognizes and begins to address sensitivity issues inherent in DNNs. Future work along this direction must ensure that module-level sensitivity debt is outweighed by system-level robustness gains. New tools for synthesis and analysis of robust learning-based modules will be key enablers of robots that account for and adapt to our environments' complexities.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Problems in Neural Network Robustness Analysis: Reachability, Verification, Minimal Adversarial Example

There has been an explosion of interest on neural network robustness analysis in recent years. DNNs are used for different purposes in different fields, so the properties that are analyzed differ, but the tools proposed are often quite similar. The goal of this section is to define several common problems of interest and highlight their similarities.

Borrowing notation from [133], let $f(x)$ be an L-layer feedforward DNN, and denote $\{0, 1, \ldots, L-1\}$ as $[L]$ and $\{x^{(0)}, x^{(1)}, \ldots, x^{(L-1)}\}$ as $x^{[L]}$. We write $f(x)$ as,

$$x^{(l+1)} = \sigma^{(l)}\left(\mathbf{W}^{(l)}x^{(l)} + b^{(l)}\right) \quad \forall l \in [L], \text{ and } f(x) = z^{(l)} = \mathbf{W}^{(L)}x^{(L)} + b^{(L)}, \quad \text{(A.1)}$$

where $x^{(l)} \in \mathbb{R}^{n^{(l)}}$, $z^{(l)} \in \mathbb{R}^{n_z^{(l)}}$, $x^{(0)} = x \in \mathbb{R}^{n_x^{(0)}}$ is the nominal input, $\mathbf{W}^{(l)} \in \mathbb{R}^{n_z^{(l)} \times n^{(l)}}$, and $b^{(l)} \in \mathbb{R}^{n_z^{(l)}}$ are the weight and bias of the l^{th} layer and $\sigma^{(l)} : \mathbb{R}^{n_z^{(l)}} \to \mathbb{R}^{n^{(l+1)}}$ is a nonlinear activation function.

Consider the optimization problem $\mathcal{O}$ (as coined by [133]),

$$\min_{x^{(0)} \in S_{\text{in}}(x^{\text{nom}})} \quad c(x^{(L)}) \tag{A.2}$$

$$\text{s.t.} \quad z^{(l)} = \mathbf{W}^{(l)} x^{(l)} + b^{(l)}, l \in [L] \tag{A.3}$$

$$x^{(l+1)} = \sigma^{(l)}(z^{(l)}), l \in [L] \tag{A.4}$$

where the network input must lie within some set around the nominal input $S_{\text{in}}(x^{\text{nom}})$ and $c : \mathbb{R}^{n^{(L)}} \to \mathbb{R}$ is a *specification* function. Various robustness analysis problems of interest are special cases of or small variations on this general formulation.

A.1 Reachability Analysis

Let $f[S_{\text{in}}(x^{\text{nom}})] = \{f(x) \mid x \in S_{\text{in}}(x^{\text{nom}})\}$ be the image of $S_{\text{in}}(x^{\text{nom}})$ under the DNN f, also called the *reachable set* from $S_{\text{in}}(x^{\text{nom}})$. The goal of reachability analysis is to compute $f[S_{\text{in}}(x^{\text{nom}})]$ exactly, though many methods instead compute a reasonably tight over-approximation $\bar{f} \supset f[S_{\text{in}}(x^{\text{nom}})]$.

For example, one can compute the tightest l_∞-ball over-approximation of $f[S_{\text{in}}(x^{\text{nom}})]$ by solving $\mathcal{O}$ for $2n^{(L)}$ different objectives,

$$c_i(x^{(L)}) = \text{sign}(i) \cdot \mathbf{e}_i^T x^{(L)} \quad \forall i \in -[n^{(L)}] \cup [n^{(L)}], \tag{A.5}$$

where $\mathbf{e}_i \in \mathbb{R}^{n^{(L)}}$ is the standard basis vector with 1 at the i^{th} element and we define,

$$\text{sign}(x) = \begin{cases} 1, \text{ if } x \geq 0 \\ -1, \text{ if } x < 0 \end{cases}. \tag{A.6}$$

A.2 (Adversarial Robustness) Verification

In classification, a neural network predicts the nominal input's class label corresponding to the largest logit, $i^* = \text{argmax}_j f_j(x^{\text{nom}})$. The goal of verification is to determine whether the network *could* output a different class label for some $x^{(0)} \in S_{\text{in}}(x^{\text{nom}})$.

One can answer this by solving $\mathcal{O}$ for $n^{(L)} - 1$ different objectives (one per non-nominal class label),

$$c_i(x^{(L)}) = (\mathbf{e}_{i^*} - \mathbf{e}_i)^T x^{(L)} \quad \forall i \in [n^{(L)}]/i^* \tag{A.7}$$

and let p_i be the cost of the solution $\forall i \in [n^{(L)}]/i^*$. If $p_i > 0 \; \forall i \in [n^{(L)}]/i^*$, then the network is verified to be *adversarially robust* for the input x^{nom} and its neighborhood $S_{\text{in}}(x^{\text{nom}})$. In other words, there is no input in $S_{\text{in}}(x^{\text{nom}})$ that could cause a non-nominal class label's logit to exceed the nominal class label's logit.

A.3 Minimal Adversarial Examples

Often in practice, the set $S_{\text{in}}(x^{\text{nom}})$ is unknown, and an algorithm is simply provided with some input x^{nom}. Nonetheless, the network robustness can be quantified in terms of the smallest perturbation that, when applied to the nominal input, would cause the network to output something other than its nominal output. The *minimal adversarial example* is the result of applying that smallest perturbation to the nominal input. Note that the question of whether that smallest perturbation is possible/likely is assumed to be handled by some other system.

To solve this problem exactly, assume something about the perturbation set's structure, e.g., it is an l_p-ball with parameter $\delta \in \mathbb{R}$,

$$S_{\text{in}}(x^{\text{nom}}, \delta) = \{x \mid ||x^{\text{nom}} - x||_p < \delta\}. \tag{A.8}$$

Then, one can modify $\mathcal{O}$ so that it returns the smallest perturbation that the network is not robust to,

$$\min_{\delta, x^{(0)} \in S_{\text{in}}(x^{\text{nom}}, \delta)} \quad \delta \tag{A.9}$$

$$\text{s.t.} \quad f_{i^*}(x^{(0)}) - f_i(x^{(0)}) > 0 \tag{A.10}$$

$$z^{(l)} = \mathbf{W}^{(l)} x^{(l)} + b^{(l)}, l \in [L] \tag{A.11}$$

$$x^{(l+1)} = \sigma^{(l)}(z^{(l)}), l \in [L] \tag{A.12}$$

[22] cleverly computes an upper bound on the smallest perturbation with a branch-and-bound version of robustness verification. At each step, the algorithm assumes some property and parameter on the input set, checks whether the network is robust to that perturbation set, and increases/decreases the parameter until convergence.

A.4 Relaxations of $\mathcal{O}$

In practice, solving $\mathcal{O}$ exactly is often computationally intractable for moderate-sized networks, which means that so are the preceding problems of interest mentioned. In particular, nonlinear activations in DNNs induce nonlinear constraints Eq. (A.4), making optimization difficult. Instead, many approaches relax those nonlinear constraints to enable efficient calculation of:

- over-approximations of reachable sets
- sound (but not complete) robustness verification results, and
- over-approximations of minimal adversarial examples.

In other words, new ideas for efficient but tight relaxations can have significant impact across the community, as they instantly yield progress in all three problems.

Bibliography

[132] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. "Efficient Neural Network Robustness Certification with General Activation Functions". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 4939–4948.

[1] Sean Hollister and Vjeran Pavic. *Skydio 2 Review: A Drone That Flies Itself*. https://www.theverge.com/2019/12/11/21009994/skydio-2-review-self-flying-autonomous-drone-camera-crash-proof-price. Accessed: 2020-04-19.

[2] Intel. *Shine Bright With Intel Light Shows*. https://www.intel.com/content/www/us/en/technology-innovation/aerial-technology-light-show.html. Accessed: 2020-04-19.

[3] Eugene Kim. *Amazon's $775 million deal for robotics company Kiva is starting to look really smart*. https://www.businessinsider.com/kiva-robots-save-money-for-amazon-2016-6. Accessed: 2020-04-20.

[4] Stephen Gossett. *12 Examples Of Rescue Robots*. https://builtin.com/robotics/rescue-robots. Accessed: 2020-04-20.

[5] T Towne. "Robot joins Michigan hospital's foodservice department". In: *Healthcare foodservice* 5.2 (1995), pp. 1–15.

[6] Elizabeth Svoboda. "Your robot surgeon will see you now". In: *Nature* 573 (2019), S110–S111.

[7] UMTRI Briefs. “Mcity Grand Opening”. In: *Research Review* 46.3 (2015).

[8] Alex Davies. *Inside the Ersatz City Where Waymo Trains Its Self-Driving Cars*. https://www.wired.com/story/google-waymo-self-driving-car-castle-testing/. Accessed: 2020-04-20.

[9] Nils J Nilsson. *Shakey the robot*. Tech. rep. SRI INTERNATIONAL MENLO PARK CA, 1984.

[10] Hassan Mujtaba. *NVIDIA Pascal GPU Analysis – An In-Depth Look at NVIDIA's Next-Gen Graphics Cards Powering GeForce, Tesla and Quadro*. https://wccftech.com/nvidia-pascal-gpu-analysis/. Accessed: 2020-04-20.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[12] Dmitry Fedyuk. *Increasing dataset sizes*. https://dmitry.ai/t/topic/198. Accessed: 2020-04-20.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] Gunter Stein. “Respect the unstable”. In: *IEEE Control systems magazine* 23.4 (2003), pp. 12–25.

[15] Michael Everett, Justin Miller, and Jonathan P. How. “Planning Beyond The Sensing Horizon Using a Learned Context”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China, 2019. URL: https://arxiv.org/pdf/1908.09171.pdf.

[16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CVPR* (2017).

[17] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[18] Michael Everett, Yu Fan Chen, and Jonathan P. How. "Collision Avoidance in Pedestrian-Rich Environments with Deep Reinforcement Learning". In: *International Journal of Robotics Research (IJRR)* (2019), in review. URL: https://arxiv.org/pdf/1910.11689.pdf.

[19] Yufan Chen, Miao Liu, Michael Everett, and Jonathan P. How. "Decentralized, Non-Communicating Multiagent Collision Avoidance with Deep Reinforcement Learning". In: *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, 2017.

[20] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. "Socially Aware Motion Planning with Deep Reinforcement Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vancouver, BC, Canada, 2017.

[21] Michael Everett*, Björn Lütjens*, and Jonathan P. How. "Certified Adversarial Robustness in Deep Reinforcement Learning". In: *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* (2020), in review. URL: https://arxiv.org/pdf/2004.06496.pdf.

[22] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S Boning, and Inderjit S Dhillon. "Towards Fast Computation of Certified Robustness for ReLU Networks". In: *International Conference on Machine Learning (ICML)*. 2018.

[23] Joshua Achiam. "Spinning Up in Deep Reinforcement Learning". In: (2018).

[24] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998.

[25] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[26] Brian Yamauchi. "Frontier-based exploration using multiple robots". In: *Proceedings of the second international conference on Autonomous agents*. ACM. 1998, pp. 47–53.

[27] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. "Information gain-based exploration using rao-blackwellized particle filters." In: *Robotics: Science and Systems*. Vol. 2. 2005, pp. 65–72.

[28] Dominik Joho, Martin Senk, and Wolfram Burgard. "Learning search heuristics for finding objects in structured environments". In: *Robotics and Autonomous Systems* 59.5 (2011), pp. 319–328.

[29] Mehdi Samadi, Thomas Kollar, and Manuela M Veloso. "Using the Web to Interactively Learn to Find Objects." In: *AAAI*. 2012, pp. 2074–2080.

[30] Thomas Kollar and Nicholas Roy. "Utilizing object-object and object-scene context when planning to find things". In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, pp. 2168–2173.

[31] Lars Kunze, Chris Burbridge, and Nick Hawes. "Bootstrapping probabilistic models of qualitative spatial relations for active visual object search". In: *AAAI Spring Symposium*. 2014, pp. 24–26.

[32] Lars Kunze, Keerthi Kumar Doreswamy, and Nick Hawes. "Using qualitative spatial relations for indirect object search". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 163–168.

[33] Malte Lorbach, Sebastian Höfer, and Oliver Brock. "Prior-assisted propagation of spatial information for object search". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE. 2014, pp. 2904–2909.

[34] Paul Vernaza and Anthony Stentz. "Learning to locate from demonstrated searches." In: *Robotics: Science and Systems*. 2014.

[35] Alper Aydemir, Andrzej Pronobis, Moritz Göbelbecker, and Patric Jensfelt. "Active visual object search in unknown environments using uncertain semantics". In: *IEEE Transactions on Robotics* 29.4 (2013), pp. 986–1002.

[36] Marc Hanheide, Moritz Göbelbecker, Graham S Horn, Andrzej Pronobis, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Charles Gretton, Richard Dearden, Miroslav Janicek, et al. "Robot task planning and explanation in open and uncertain worlds". In: *Artificial Intelligence* 247 (2017), pp. 119–150.

[37] Samarth Brahmbhatt and James Hays. "Deepnav: Learning to navigate large cities". In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* 2017, pp. 3087–3096.

[38] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. "Building generalizable agents with a realistic and rich 3D environment". In: *arXiv preprint arXiv:1801.02209* (2018).

[39] Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A Knepper, and Yoav Artzi. "Following High-level Navigation Instructions on a Simulated Quadcopter with Imitation Learning". In: *arXiv preprint arXiv:1806.00047* (2018).

[40] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. "Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation*. 2017.

[41] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. "Visual semantic planning using deep successor representations". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 483–492.

[42] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. "IQA: Visual question answering in interactive environments". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4089–4098.

[43] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. "Cognitive mapping and planning for visual navigation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2616–2625.

[44] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. "Embodied Question Answering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[46] Andrzej Pronobis and Rajesh PN Rao. "Learning deep generative spatial models for mobile robots". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 755–762.

[47] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. "On evaluation of embodied navigation agents". In: *arXiv preprint arXiv:1807.06757* (2018).

[48] C. Richter, J. Ware, and N. Roy. "High-speed autonomous navigation of unknown environments using learned probabilities of collision". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 6114–6121. DOI: 10.1109/ICRA.2014.6907760.

[49] Charles Richter and Nicholas Roy. "Learning to Plan for Visibility in Navigation of Unknown Environments". In: *2016 International Symposium on Experimental Robotics*. Ed. by Dana Kulić, Yoshihiko Nakamura, Oussama Khatib, and Gentiane Venture. Cham: Springer International Publishing, 2017, pp. 387–398.

[50] Charles Richter, William Vega-Brown, and Nicholas Roy. "Bayesian Learning for Safe High-Speed Navigation in Unknown Environments". In: *Robotics Research: Volume 2*. Ed. by Antonio Bicchi and Wolfram Burgard. Cham: Springer International Publishing, 2018, pp. 325–341. DOI: 10.1007/978-3-319-60916-4_19. URL: https://doi.org/10.1007/978-3-319-60916-4_19.

[51] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. "Learning heuristic search via imitation". In: *arXiv preprint arXiv:1707.03034* (2017).

[52] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. "Building Generalizable Agents with a Realistic and Rich 3D Environment". In: *CoRR* abs/1801.02209 (2018). arXiv: 1801.02209. URL: http://arxiv.org/abs/1801.02209.

[53] Christopher Hesse. *Image-to-Image Translation in Tensorflow*. https://affinelayer.com/pix2pix/. Accessed: 2018-08-28.

[54] Microsoft. *Bing Maps*. https://www.bing.com/maps. Accessed: 2019-03-01.

[55] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn". In: *IEEE transactions on pattern analysis and machine intelligence* (2018).

[56] Zhang Xuan and Filliat David. *Real-time voxel based 3D semantic mapping with a hand held RGB-D camera*. https://github.com/floatlazer/semantic_slam. 2018.

[57] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[58] Lucas Willems Maxime Chevalier-Boisvert. *Minimalistic Gridworld Environment for OpenAI Gym*. https://github.com/maximecb/gym-minigrid. 2018.

[59] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[60] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533.

[61] Clearpath Robotics. *Jackal Unmanned Ground Vehicle*. https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/. [Online; accessed 22-Feb-2018].

[62] Abhinav Valada, Rohit Mohan, and Wolfram Burgard. "Self-Supervised Model Adaptation for Multimodal Semantic Segmentation". In: *International Journal of Computer Vision (IJCV)* (2019). Special Issue: Deep Learning for Robotic Vision. DOI: 10.1007/s11263-019-01188-y.

[63] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. "A navigation system for robots operating in crowded urban environments". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3225–3232.

[64] P. Trautman and A. Krause. "Unfreezing the robot: Navigation in dense, interacting crowds". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2010, pp. 797–803. DOI: 10.1109/IROS.2010.5654369.

[65] J. Snape, J. Van den Berg, S. J. Guy, and D. Manocha. "The hybrid reciprocal velocity obstacle". In: *IEEE Transactions on Robotics* 27.4 (Aug. 2011), pp. 696–706. DOI: 10.1109/TRO.2011.2120810.

[66] G. Ferrer, A. Garrell, and A. Sanfeliu. "Social-aware robot navigation in urban environments". In: *2013 European Conference on Mobile Robots (ECMR)*. Sept. 2013, pp. 331–336. DOI: 10.1109/ECMR.2013.6698863.

[67] Jur Van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. "Reciprocal n-body collision avoidance". en. In: *Robotics Research*. Springer Tracts in Advanced Robotics 70. Springer Berlin Heidelberg, 2011, pp. 3–19. DOI: 10.1007/978-3-642-19457-3_1.

[68] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. "Optimal reciprocal collision avoidance for multiple non-holonomic robots". In: *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 203–216.

[69] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. "Socially compliant mobile robot navigation via inverse reinforcement learning". en. In: *The International Journal of Robotics Research* (Jan. 2016). DOI: 10.1177/0278364915619772.

[70] P. Trautman, J. Ma, R. M. Murray, and A. Krause. "Robot navigation in dense human crowds: the case for cooperation". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 2153–2160. DOI: 10.1109/ICRA.2013.6630866.

[71] Markus Kuderer, Henrik Kretzschmar, Christoph Sprunk, and Wolfram Burgard. "Feature-based prediction of trajectories for socially compliant navigation". In: *Robotics:Science and Systems*. 2012.

[72] Pinxin Long, Tingxiang Fanl, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6252–6259.

[73] Lei Tai, Giuseppe Paolo, and Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 31–36.

[74] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[75] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[76] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[77] James B Rawlings. "Tutorial overview of model predictive control". In: *IEEE control systems magazine* 20.3 (2000), pp. 38–52.

[78] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics Automation Magazine* 4.1 (Mar. 1997), pp. 23–33. DOI: 10.1109/100.580977.

[79] G. Ferrer, A. Garrell, and A. Sanfeliu. "Robot Companion: A Social-Force Based Approach with Human Awareness-Navigation in Crowded Environments". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 1688–1694. DOI: 10.1109/IROS.2013.6696576.

[80] M. Phillips and M. Likhachev. "SIPP: Safe interval path planning for dynamic environments". In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. May 2011, pp. 5628–5635. DOI: 10.1109/ICRA.2011.5980306.

[81] Georges S. Aoude, Brandon D. Luders, Joshua M. Joseph, Nicholas Roy, and Jonathan P. How. "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns". en. In: *Autonomous Robots* 35.1 (May 2013), pp. 51–76. DOI: 10.1007/s10514-013-9334-3.

[82] Christoforos I Mavrogiannis and Ross A Knepper. "Multi-agent path topology in support of socially competent navigation planning". In: *The International Journal of Robotics Research* 38.2-3 (2019), pp. 338–356. DOI: 10.1177/0278364918781016. eprint: https://doi.org/10.1177/0278364918781016. URL: https://doi.org/10.1177/0278364918781016.

[83] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[84] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. "Reinforcement Learning thorugh Asynchronous Advantage Actor-Critic on a GPU". In: *ICLR*. 2017.

[85] Shayegan Omidshafiei, Dong-Ki Kim, Jason Pazis, and Jonathan P How. "Crossmodal Attentive Skill Learner". In: *arXiv preprint arXiv:1711.10314* (2017).

[86] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[87] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *arXiv preprint arXiv:1802.09477* (2018).

[88] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[89] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[90] Beomjoon Kim and Joelle Pineau. "Socially adaptive path planning in human environments using inverse reinforcement learning". en. In: *International Journal of Social Robotics* 8.1 (June 2015), pp. 51–66. DOI: 10.1007/s12369-015-0310-2.

[91] Mark Pfeiffer, Ulrich Schwesinger, Hannes Sommer, Enric Galceran, and Roland Siegwart. "Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2096–2101.

[92] Javier Garcıa and Fernando Fernández. "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[93] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 961–971.

[94] Christopher Olah. *Understanding LSTM Networks*. COURSERA: Neural Networks for Machine Learning. 2015.

[95] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[96] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[97] Universe Lau. *rl-collision-avoidance*. `https://github.com/Acmece/rl-collision-avoidance`. [Online; accessed 10-Sep-2019]. 2019.

[98] Intel. *Intel Drones Light Up the Sky*. `https://www.intel.com/content/www/us/en/technology-innovation/aerial-technology-light-show.html`. [Online; accessed 4-Sep-2019]. 2019.

[99] Airbus. *Airbus Commercial Aircraft formation flight: 50-year anniversary*. `https://www.youtube.com/watch?v=JS6w-DXiZpk`. [Online; accessed 4-Sep-2019]. 2019.

[100] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. "Robocup: The robot world cup initiative". In: (1995).

[101] Pixar. *Finding Nemo (School of Fish Scene)*. `https://www.youtube.com/watch?v=Le13by2WM70`. [Online; accessed 4-Sep-2019]. 2003.

[102] Shayegan Omidshafiei, Ali akbar Agha-mohammadi, Yu Fan Chen, N. Kemal Ure, Jonathan How, John Vian, and Rajeev Surati. "MAR-CPS: Measurable Augmented Reality for Prototyping Cyber-Physical Systems". In: 2015.

[103] Trevor Campbell, Miao Liu, Brian Kulis, Jonathan P How, and Lawrence Carin. "Dynamic clustering via asymptotics of the dependent Dirichlet process mixture". In: *Advances in Neural Information Processing Systems*. 2013, pp. 449–457.

[104] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[105] J. Miller, A. Hasfura, S. Y. Liu, and J. P. How. "Dynamic Arrival Rate Estimation for Campus Mobility On Demand Network Graphs". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 2285–2292. DOI: 10.1109/IROS.2016.7759357.

[106] Michael Everett. "Robot Designed for Socially Acceptable Navigation". Master Thesis. Cambridge, MA, USA: MIT, June 2017.

[107] S. Gu, E. Holly, T. Lillicrap, and S. Levine. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[108] Tingxiang Fan, Xinjing Cheng, Jia Pan, Pinxin Long, Wenxi Liu, Ruigang Yang, and Dinesh Manocha. "Getting Robots Unfrozen and Unlost in Dense Pedestrian Crowds". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1178–1185.

[109] Michael Everett, Yu Fan Chen, and Jonathan P. How. "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2018.

[110] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *International Conference on Learning Representations (ICLR)*. 2014.

[111] Naveed Akhtar and Ajmal S. Mian. "Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey". In: *IEEE Access* 6 (2018), pp. 14410–14430.

[112] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. "Adversarial Examples: Attacks and Defenses for Deep Learning". In: *IEEE transactions on neural networks and learning systems* (2019).

[113] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *International Conference on Learning Representation (ICLR) (Workshop)*. 2017.

[114] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1528–1540.

[115] Tencent Keen Security Lab. *Experimental Security Research of Tesla Autopilot*. Mar. 2019. URL: `https://keenlab.tencent.com/en/whitepapers/\\Experimental\_Security\_Research\_of\_Tesla\_Autopilot.pdf`.

[116] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. "Adversarially robust policy learning: Active construction of physically-plausible perturbations". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3932–3939.

[117] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. "EPOpt: Learning Robust Neural Network Policies Using Model Ensembles". In: *International Conference on Learning Representations (ICLR)*. 2017.

[118] Fabio Muratore, Felix Treede, Michael Gienger, and Jan Peters. "Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment". In: *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*. 2018, pp. 700–713.

[119] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. "Robust Adversarial Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 2817–2826.

[120] Jun Morimoto and Kenji Doya. "Robust reinforcement learning". In: *Neural computation* 17.2 (2005), pp. 335–359.

[121] Rüdiger Ehlers. "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks". In: *ATVA*. 2017.

[122] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. 2017, pp. 97–117.

[123] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. "Safety Verification of Deep Neural Networks". In: *Computer Aided Verification*. Ed. by Rupak Majumdar and Viktor Kunčak. Cham: Springer International Publishing, 2017, pp. 3–29.

[124] Alessio Lomuscio and Lalit Maganti. "An approach to reachability analysis for feed-forward ReLU neural networks". In: *CoRR* abs/1706.07351 (2017). arXiv: 1706.07351. URL: http://arxiv.org/abs/1706.07351.

[125] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *International Conference on Learning Representations (ICLR)*. 2019.

[126] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 3–18.

[127] Eric Wong and J. Zico Kolter. "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope". In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. 2018, pp. 5283–5292.

[128] Chaoqun Wang, Jiyu Cheng, Jiankun Wang, Xintong Li, and Max Q-H Meng. "Efficient Object Search With Belief Road Map Using Mobile Robot". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3081–3088.

[129] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. "Training verified learners with learned verifiers". In: *arXiv preprint arXiv:1805.10265* (2018).

[130] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. "Fast and effective robustness certification". In: *Advances in Neural Information Processing Systems*. 2018, pp. 10802–10813.

[131] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. "An abstract domain for certifying neural networks". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–30.

[133] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. "A convex relaxation barrier to tight robustness verification of neural networks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 9832–9842.

[134] Björn Lütjens, Michael Everett, and Jonathan P. How. "Certified Adversarial Robustness for Deep Reinforcement Learning". In: *2019 Conference on Robot Learning (CoRL)*. Osaka, Japan, 2019. URL: `https://arxiv.org/pdf/1910.12908.pdf`.

[135] Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala Al-Fuqaha, Dinh Thai Hoang, and Dusit Niyato. "Challenges and Countermeasures for Adversarial Attacks on Deep Reinforcement Learning". In: *arXiv preprint arXiv:2001.09684* (2020).

[136] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations (ICLR)*. 2015.

[137] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. "Adversarial Attacks on Neural Network Policies". In: (2017).

[138] Vahid Behzadan and Arslan Munir. "Vulnerability of deep reinforcement learning to policy induction attacks". In: *International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*. Springer. 2017, pp. 262–275.

[139] William Uther and Manuela Veloso. *Adversarial Reinforcement Learning*. Tech. rep. In Proceedings of the AAAI Fall Symposium on Model Directed Autonomous Systems, 1997.

[140] Adam Gleave, Michael Dennis, Neel Kant, Cody Wild, Sergey Levine, and Stuart Russell. "Adversarial policies: Attacking deep reinforcement learning". In: (2020).

[141] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[142] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale". In: *International Conference on Learning Representations (ICLR)*. 2017.

[143] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations (ICLR)*. 2018.

[144] Jernej Kos and Dawn Song. "Delving into adversarial attacks on deep policies". In: *International Conference on Learning Representations (ICLR) (Workshop)*. 2017.

[145] Matthew Mirman, Marc Fischer, and Martin Vechev. *Distilled Agent DQN for Provable Adversarial Robustness*. 2019. URL: https://openreview.net/forum?id=ryeAy3AqYm.

[146] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks". In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 582–597.

[147] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. "Ensemble Adversarial Training: Attacks and Defenses". In: *International Conference on Learning Representations (ICLR)*. 2018.

[148] Weilin Xu, David Evans, and Yanjun Qi. "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks". In: *Network and Distributed Systems Security Symposium (NDSS)*. The Internet Society, 2018.

[149] Nicholas Carlini and David Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. AISec '17. Dallas, Texas, USA: ACM, 2017, pp. 3–14.

[150] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. "Adversarial Example Defenses: Ensembles of Weak Defenses Are Not Strong". In: *Proceedings of the 11th USENIX Conference on Offensive Technologies*. WOOT'17. Vancouver, BC, Canada: USENIX Association, 2017, pp. 15–15.

[151] Anish Athalye, Nicholas Carlini, and David Wagner. "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 274–283.

[152] Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron van den Oord. "Adversarial Risk and the Dangers of Evaluating Against Weak Attacks". In: *Proceedings of the 35th International Conference on Machine Learn-*

ing (ICML). Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 5025–5034.

[153] Javier García and Fernando Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research* 16 (2015), pp. 1437–1480.

[154] Matthias Heger. "Consideration of Risk in Reinforcement Learning". In: *Machine Learning Proceedings 1994*. Ed. by William W. Cohen and Haym Hirsh. San Francisco (CA): Morgan Kaufmann, 1994, pp. 105 –111.

[155] Aviv Tamar. "Risk-sensitive and Efficient Reinforcement Learning Algorithms". PhD thesis. Technion - Israel Institute of Technology, Faculty of Electrical Engineering, 2015.

[156] Peter Geibel. "Risk-Sensitive Approaches for Reinforcement Learning". PhD thesis. University of Osnabrück, 2006.

[157] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature*. Vol. 518. Nature Publishing Group, a division of Macmillan Publishers Limited., 2015.

[158] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[159] Afonso S Bandeira. "A note on probably certifiably correct algorithms". In: *Comptes Rendus Mathematique* 354.3 (2016), pp. 329–333.

[160] Heng Yang, Jingnan Shi, and Luca Carlone. "TEASER: Fast and Certifiable Point Cloud Registration". In: (2020). arXiv: `2001.07715 [cs.RO]`. URL: `https://github.com/MIT-SPARK/TEASER-plusplus`.

[161] Michael Everett and Jonathan How. *Gym: Collision Avoidance*. `https://github.com/mit-acl/gym-collision-avoidance`. 2020.

[162] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[163] Jur P. van den Berg, Stephen J. Guy, Ming C. Lin, and Dinesh Manocha. "Reciprocal n-Body Collision Avoidance". In: *International Symposium on Robotics Research (ISRR)*. 2009.

[164] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems (NeurIPS) 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 2951–2959.

[165] Andrew G Barto, Richard S Sutton, and Charles W Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[166] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. *Stable Baselines*. `https://github.com/hill-a/stable-baselines`. 2018.

[167] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. *CROWN-IBP: Towards Stable and Efficient Training of Verifiably Robust Neural Networks*. `https://github.com/huanzhang12/CROWN-IBP`. 2020.

[168] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[169] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *arXiv preprint arXiv:1601.06759* (2016).

[170] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. "Toward multimodal image-to-image translation". In: *Advances in neural information processing systems*. 2017, pp. 465–476.

[171] Bruno Brito, Boaz Floor, Laura Ferranti, and Javier Alonso-Mora. "Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4459–4466.

[172] Samaneh Hosseini Semnani, Hugh Liu, Michael Everett, Anton de Ruiter, and Jonathan P How. "Multi-agent Motion Planning for Dense and Dynamic Environments via Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters (RA-L)* (2020). URL: https://arxiv.org/pdf/2001.06627.pdf.

[173] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. "Fully distributed multi-robot collision avoidance via deep reinforcement learning for safe and efficient navigation in complex scenarios". In: *arXiv preprint arXiv:1808.03841* (2018).