# ROBOT NAVIGATION IN CROWDED DYNAMIC SCENES

---

A Dissertation
Submitted to
the Temple University Graduate Board

---

In Partial Fulfillment
of the Requirements for the Degree
DOCTOR OF PHILOSOPHY

---

by
Zhanteng Xie
August 2024

Examining Committee Members:

Philip Dames, Advisory Chair, Mechanical Engineering
Damoon Soudbakhsh, Mechanical Engineering
Daniel A. Jacobs, Mechanical Engineering
Xuesu Xiao, George Mason University
Iyad Obeid, External Reader, Electrical and Computer Engineering

©
Copyright
2024

by

Zhanteng Xie

All Rights Reserved

## ABSTRACT

Autonomous mobile robots are beginning to try to help us provide different delivery services in people's lives, such as delivering medicines in hospitals, delivering goods in warehouses, and delivering food in restaurants. To realize this vision, robots need to navigate autonomously and efficiently through complex, crowded, and dynamic environments filled with static obstacles, such as tables and chairs, as well as people and/or other robots, and to achieve this using the computational resources available onboard a mobile robot. This dissertation improves the state-of-the-art in autonomous navigation by developing learning-based algorithms to model the environment around the robot, predict changes in the environment, and control the robot, all of which can run onboard a mobile robot in real time. Specifically, this dissertation first proposes a set of specialized preprocessed data representations to extract and encode useful high-level information about crowded dynamic environments from raw sensor data (*i.e.,* a short history of lidar data, kinematic data about nearby pedestrians, and a sub-goal that leads the robots towards its final destination). Then, using these combined preprocessed data representations, this dissertation proposes a novel crowd-aware navigation control policy that can balance collision avoidance and speed in crowded dynamic scenes by designing a velocity obstacle-based reward function that is used to train the robot leveraging deep reinforcement learning techniques. This dissertation then proposes a series of hardware-friendly prediction algorithms, based on variational autoencoder networks, to predict a distribution of possible future states in dynamic scenes by exploiting the kinematics and dynamics of the robot and its surrounding objects. Furthermore, this dissertation proposes a novel predictive uncertainty-aware navigation framework to improve the safety performance of current existing control policies by incorporating the

output of the proposed stochastic environment prediction algorithms into general navigation frameworks. Many different collected real-world datasets as well as a series of 3D simulation experiments and hardware experiments are used to demonstrate the effectiveness of these proposed novel learning-based prediction and control algorithms. The new algorithms outperform other state-of-the-art algorithms in terms of collision avoidance, robot speed, and prediction accuracy across a range of environments, crowd densities, and robot models. It is believed that all the work included in this dissertation will promote the development of autonomous navigation for modern mobile robots, provide highly innovative solutions to the open problem of autonomous navigation in crowded dynamic scenes, and make our daily lives more convenient and efficient.

Dedicated to my family.

## ACKNOWLEDGMENTS

Time flies, and in the blink of an eye, my five-year Ph.D. career has passed.

Firstly, I would like to thank my Ph.D. advisor, Prof. Philip Dames, who gave me a wonderful opportunity to study and work with him as his second Ph.D. student. Over the past five years, Prof. Dames has been meticulously, patiently, and attentively guiding me on how to do research, how to write papers, how to give presentations, how to review papers, how to teach, and how to tutor students. He gave me the greatest research freedom and learning space, helping me to become an independent researcher step by step. When I encountered bottlenecks in my research, he always encouraged me, gave me enough time, and used his clever wisdom to give me useful suggestions to help me overcome these difficulties. When I encountered difficulties in life, he also gave me the greatest support and tried his best to help me. I am grateful to Prof. Dames for teaching me to fish instead of giving me the fish. I am also grateful to have learned about his philosophy of slow science research. All in all, I can confidently say that Prof. Dames is the best advisor I have ever had in my life.

Secondly, I would like to thank my dissertation committee members, Dr. Damoon Soudbakhsh, Dr. Daniel A. Jacobs, Dr. Xuesu Xiao, and Dr. Iyad Obeid for taking the time to serve on my committee, providing helpful feedback on my dissertation, and helping me graduate. I was privileged to take classes from Dr. Soudbakhsh and Dr. Jacobs during my Ph.D. period. Their engaging teaching courses laid the foundation for my robotics research. I also had the pleasure of discussing research and career with Dr. Xiao and receiving his helpful guidance at different times. I would also like to pay tribute to Dr. Obeid, who reached out at the last minute to be an external reader for my dissertation.

Thirdly, I would like to thank all the team members who currently or formerly worked in the TRAIL lab and the RoboSNAP group. It was a great time working with the TRAIL lab members and the RoboSNAP members, especially conducting human-robot psychology experiments with the RoboSNAP members in the SERC Lobby and the Engineering Building (special thanks to Lindsay Ouellette and Dr. Donald Hantula). Furthermore, I would like to especially thank my lab member and roommate Pujie Xin, who has left me with many interesting memories in research and daily life, and made my Ph.D. life more colorful over the past four years. I would also like to thank my former lab member Jun Chen, who provided great help in my daily life during my first two years at Temple University. Additionally, I would like to thank my young friend Renjie Song for leading me in chasing the Northern Lights and organizing several interesting and meaningful road trips for our Temple Mechanical Team.

Finally, I would like to thank my family, especially my parents, for their sincere understanding and emotional support. Without their support, I would not have been able to complete my Ph.D. studies. I know they have been through a lot on their own in the five years of my absence. But they are still willing to be my warmest harbor, a place I can return to no matter how far I go. I still remember what my father said when I went abroad to study for my Ph.D. Degree: "You are now like an orange tree seedling. It takes 5 years for it to grow up and bear fruit." At this moment, I can proudly tell my father that this five-year-old orange tree has already blossomed and borne fruit.

## TABLE OF CONTENTS

Page

ABSTRACT ..... iii

DEDICATION ..... v

ACKNOWLEDGMENTS ..... vi

LIST OF TABLES ..... xiv

LIST OF FIGURES ..... xv

CHAPTER

1. INTRODUCTION ..... 1

2. BACKGROUND ..... 6

2.1. Deep Learning Introduction ..... 6

2.1.1. Fully Connected Neural Network ..... 8

2.1.2. Convolutional Neural Network ..... 9

2.1.3. Recurrent Neural Network ..... 10

2.2. Primary Learning Techniques ..... 10

2.2.1. Supervised Learning ..... 12

2.2.2. Unupervised Learning ..... 12

2.2.3. Reinforcement Learning ..... 14

2.3. Deep Neural Network Optimization ..... 16

Page


2.3.1. Network Pruning ..... 17

2.3.2. Weight Quantization and Sharing ..... 17

2.3.3. Knowledge Distillation ..... 18

2.3.4. Customized Network Design ..... 18

2.4. Summary ..... 19

3. LITERATURE REVIEW ..... 20

3.1. Overview ..... 20

3.2. Environmental Data Processing ..... 25

3.2.1. Environmental Perception and Representation ..... 25

3.2.2. Environmental Prediction ..... 26

3.3. Control Policy Design ..... 29

3.3.1. Model-based Approaches ..... 29

3.3.2. Supervised Learning-based Approaches ..... 30

3.3.3. Reinforcement Learning-based Approaches ..... 31

3.4. Comparative Analysis ..... 33

4. ENVIRONMENTAL MODELING AND REPRESENTATION ..... 36

4.1. Problem Formulation ..... 37

4.2. Preprocessed Data Representation ..... 38

4.2.1. System Overview ..... 38

Page

4.2.2. Pedestrian Data Preprocessing ..... 40

4.2.3. Lidar Data Preprocessing ..... 43

4.2.4. Goal Position ..... 45

4.3. Experiments and Results ..... 46

4.3.1. Simulation Configuration ..... 47

4.3.2. Data Collection ..... 49

4.3.3. Training Results ..... 50

4.3.4. Navigation Results ..... 51

4.4. Conclusion ..... 56

5. CROWD-AWARE CONTROL POLICY ..... 57

5.1. Problem Formulation ..... 58

5.2. DRL-based Control Policy ..... 59

5.2.1. Observation Space ..... 59

5.2.2. Action Space ..... 61

5.2.3. Network Architecture ..... 61

5.2.4. Reward Function ..... 63

5.2.5. Deep Reinforcement Learning Algorithm ..... 67

5.3. Crowded Dynamic Environments ..... 69

5.3.1. Experimental Setup ..... 69

Page


5.3.2. Training Procedure ..... 73

5.3.3. Simulation Results ..... 74

5.3.4. Hardware Results ..... 82

5.4. Highly Constrained Environments ..... 89

5.4.1. Experimental Setup ..... 90

5.4.2. Deployment Adjustment ..... 91

5.4.3. Results ..... 92

5.4.4. Discussion ..... 97

5.5. Lessons Learned ..... 97

5.5.1. Data Representations ..... 98

5.5.2. Computational Constraints ..... 99

5.5.3. Training Methodology ..... 99

5.6. Conclusion ..... 102

6. ENVIRONMENTAL PREDICTION AND UNCERTAINTY-AWARE CONTROL POLICY ..... 104

6.1. Problem Formulation ..... 105

6.1.1. Image-Based Prediction ..... 107

6.1.2. SCOPE++ ..... 108

6.1.3. Robot Motion Compensation $k(\cdot)$ ..... 109

6.1.4. Dynamic Object Prediction $h(\cdot)$ ..... 112

Page


6.1.5. Static Object Segmentation $g(\cdot)$ ..... 113

6.1.6. VAE Predictor ..... 114

6.2. Software Optimization ..... 116

6.2.1. Resource Utilization ..... 117

6.2.2. Knowledge Distillation ..... 118

6.2.3. Uncertainty Quantification ..... 119

6.3. OGM Prediction Experiments and Results ..... 123

6.3.1. Datasets ..... 123

6.3.2. OGM Prediction ..... 125

6.3.3. Computational Resource Utilization ..... 136

6.3.4. Uncertainty Characterization ..... 138

6.4. Uncertainty-Aware Navigation ..... 143

6.4.1. Baselines and Evaluation Metrics ..... 146

6.4.2. Experiment Setup ..... 147

6.4.3. Simulation Results ..... 150

6.4.4. Hardware Results ..... 152

6.5. Conclusion ..... 154

7. CONCLUSION ..... 156

7.1. Contributions ..... 156

Page

7.2. Future Work ..... 158

7.2.1. Socially-Aware Control Policy ..... 158

7.2.2. A Lightweight Unified Framework For Robot Navigation .... 159

7.3. Concluding Remarks ..... 160

REFERENCES ..... 162

## LIST OF TABLES

Table Page

3.1. Summary of the different categories of robot navigation methods..... 33

4.1. Training results on the robot navigation datasets..... 51

4.2. Navigation results in different simulated environments. ..... 52

5.1. Ablation experiment results with different fusion structures..... 62

5.2. Navigation results at different crowd densities..... 75

5.3. Navigation results at different unseen environments with 25 pedestrians. ..... 77

5.4. Navigation results at different unseen environments with 35 pedestrians. ..... 78

5.5. Navigation results at BARN simulation environments..... 94

6.1. Resource utilization percentage of SCOPE++ predictor on the embedded device. 117

6.2. Inference speed and model size of different OGM predictors..... 136

6.3. Navigation results in the Lobby world at different crowd densities. ..... 149

## LIST OF FIGURES

Figure Page

1.1. Examples of autonomous robot delivery services. ..... 2

1.2. Basic architecture for autonomous navigation..... 3

1.3. Dissertation outline. ..... 5

2.1. An illustration of the deep learning structure. ..... 7

2.2. An illustration of different activation functions. ..... 8

2.3. Three primary deep learning techniques. ..... 11

3.1. Illustration of the robot navigation problem..... 21

3.2. Basic frameworks of the robot navigation stack. ..... 23

3.3. Basic structure of learning-based control approaches..... 24

4.1. Architecture of the proposed CNN deep neural network..... 40

4.2. Illustration of the proposed novel pedestrian kinematic maps and lidar historical map..... 41

4.3. Validation loss curves of different lidar data configurations. ..... 46

4.4. Preprocessing step for the sub-goal position. ..... 47

4.5. Gazebo simulation environments. ..... 48

4.6. Validation loss curves during the training process. ..... 52

4.7. Robot reactions to moving pedestrians using different control policies. ..... 53

5.1. Overall system architecture of the proposed DRL-VO control policy. ..... 60

5.2. The pictogram for velocity obstacle $VO_{A,B}$ (red cone), collision cone $CC_{A,B}$ (light blue cone), and special occupancy $SO_{A,B}$ (blue circle). ..... 65

5.3. Gazebo simulation environments. ..... 71

5.4. Floorplans of real-world environments. ..... 73

Figure Page

5.5. Histograms of the output velocities for different control policies. ..... 80

5.6. Robot reactions to moving pedestrians in the indoor hallway with the low crowd density at different times. ..... 82

5.7. Robot reactions to moving pedestrians in the indoor hallway with the medium crowd density at different times. ..... 83

5.8. Robot reactions to moving pedestrians in the indoor hallway with the high crowd density at different times. ..... 85

5.9. Robot reactions to moving pedestrians in the indoor lobby with the high crowd density at different times. ..... 87

5.10. Robot reactions to moving pedestrians in the outdoor hallway with the high crowd density at different times. ..... 88

5.11. BARN simulation environment examples. ..... 89

5.12. BARN real-world testing environments. ..... 91

5.13. Successful navigation behavior of the Jackal robot in the wide “S” shaped environment at different times. ..... 95

5.14. Typical failure cases in the pointed/narrow “S” shaped environment. ..... 96

6.1. A simple illustration of the OGM prediction problem. ..... 106

6.2. System architectures of the proposed SCOPE++ predictor, SCOPE predictor, and its software-accelerated SO-SCOPE predictor. ..... 110

6.3. Uncertainty quantification visualization of predicted OGM cell bins for different prediction timesteps. ..... 120

6.4. Illustration of the OGM dataset collection from three different robots. ..... 124

6.5. Evaluation pipeline for calculating OSPA error on predicted OGMs. ..... 127

6.6. Two Prediction showcases of seven OGM predictors tested on the OGM-Turtlebot2 dataset over the prediction horizon. ..... 129

6.7. Average WMSE of 10 different prediction time steps for all tested methods on different datasets. ..... 131

Figure Page

6.8. Average SSIM of 10 different prediction time steps for all tested methods on different datasets. ..... 133

6.9. Average OSPA of 10 different prediction time steps for all tested methods on different datasets. ..... 135

6.10. Inference speed and memory usage of SCOPE and SO-SCOPE predictors with different number of samples. ..... 137

6.11. A diverse set of prediction samples of our SCOPE++, SCOPE, and SO-SCOPE predictors tested on the OGM-Turtlebot2 dataset at the 5th prediction timestep. 140

6.12. Average entropy of our SCOPE, SCOPE++, and SO-SCOPE predictors at 5th prediction time step on OGM-Turtlebot2 dataset. ..... 141

6.13. Cell counts per bin in the SCOPE, SCOPE++, and SO-SCOPE predictors. ..... 142

6.14. System architectures of the proposed SCOPE-based and SO-SCOPE-based predictive uncertainty-aware navigation planners. ..... 145

6.15. Robot navigation test environments in the simulation and real world. ..... 148

6.16. Robot reactions and their corresponding costmaps generated by different DWA-based control policies in the simulated lobby environment. ..... 151

6.17. Robot deployed with DWA-SCOPE-PU reactions to moving pedestrians in the indoor hallway with the high crowd density at different times. ..... 153

6.18. Robot deployed with DRL-VO-SO-SCOPE-PU reactions to moving pedestrians in the indoor hallway with the high crowd density at different times. ..... 154

# CHAPTER 1
# INTRODUCTION

Mobile robots are beginning to enter people's lives and try to help us provide last-mile delivery services in different human scenarios. For example, delivering sterile supplies and injection medicines to patients in hospitals [1], delivering materials to various packaging workstations in warehouses [2], delivering groceries to local residents in grocery stores [3], and delivering delicious food to customers in restaurants [4], as shown in Fig. 1.1. Note that these different application scenarios are complex, dynamic, and human-filled environments, but current commercial robots are limited to working in slow/conservative motion, relatively simple, and sparsely populated conditions. There is still a long way to go to integrate mobile robots into the crowded dynamic space in a more complete manner, where all these application tasks have time limits and require mobile robots to navigate autonomously, efficiently, reliably, and even socially to destinations through a partially known space filled with moving crowds and other static obstacles. This dissertation aims to address some of the technical challenges of autonomous robot navigation in crowded dynamic scenes. While static obstacles such as tables, boxes, and chairs that can be pre-mapped can be easily and safely avoided, the main challenge lies in autonomously navigating through crowds of pedestrians and other service robots, all of which are dynamically moving, cannot be pre-mapped, and have no clear way to communicate and interact.

To enable autonomous navigation, a mobile robot given a goal position first perceives its surroundings using its sensors such as lidars and cameras, estimates its own state (*i.e.,* localizes its position relative to the environments), predicts what is going to happen, models the states of the environment and represents them, reasons and plans a nominal path using

(a) Hospital

(b) Warehouse

(c) Grocery store

(d) Restaurant

Figure 1.1. Examples of autonomous robot delivery services.

the prior knowledge (*e.g.,* goal, map, and environment obstacles), calculate real-time control commands according to the processed environmental states, and finally execute these control commands to move to the goal position. The basic architecture of the autonomous navigation system is shown in Fig. 1.2. Since autonomous navigation is a system-level problem, this dissertation uses off-the-shelf solutions for the other parts (*i.e.,* perception, localization, planning, and actuation), and focuses on the core issues in environmental prediction, environmental modeling and representation, and the control policy design, as shown in red font in the Fig. 1.2. Specifically, these key issues include how to process perceived sensor data and predict future environmental states (*e.g.,* pedestrian prediction, scene geometry prediction, and human-robot interaction); how to model these environmen-

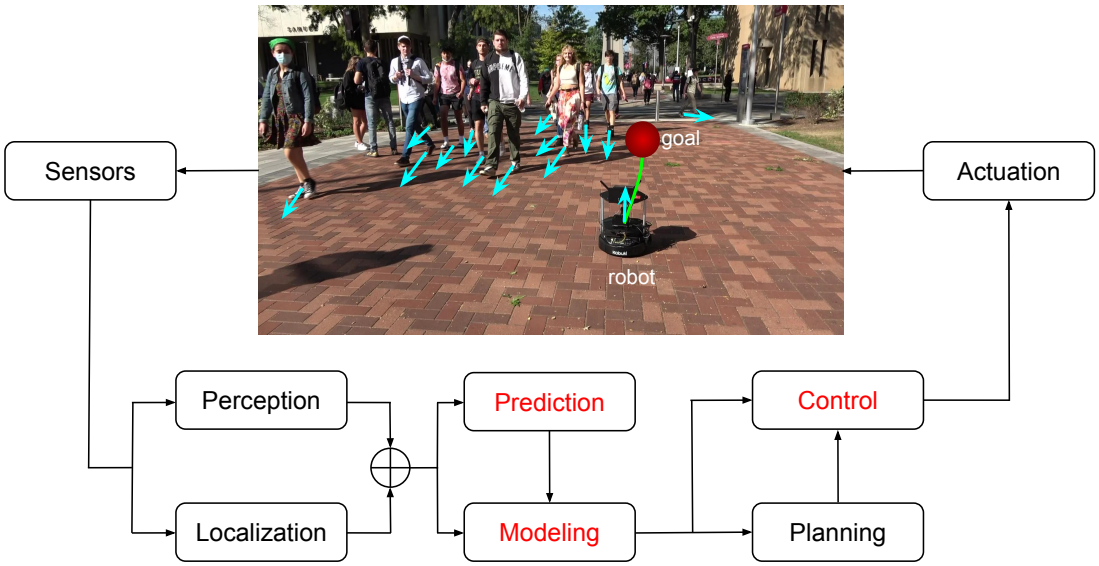



Figure 1.2. Basic architecture for autonomous navigation.

tal states and represent them in an efficient and concise abstraction, and how to design robust control policies that yield autonomous, efficient, reliable, and even socially-aware navigation in crowded dynamic environments.

Current traditional navigation systems [5] are heuristic-based and optimization-based approaches that require explicitly defined rules, manual tuning of model parameters, and suffer from the curse of dimensionality. However, the complex, dynamic, and implicit human-robot and human-human interaction rules are difficult to define explicitly and mathematically. The high dimensionality issues in crowded dynamic scenes also make traditional navigation approaches unable to solve them. To overcome the aforementioned limitations, this dissertation explores learning-based approaches, as they can implicitly model and represent the complex human-robot and human-human interactions by using neural networks and address the dimensionality issues by exploiting the powerful approximation capabilities of neural networks and utilizing a large amount of data. Specifically, this dissertation aims to: 1) design novel environmental representations that are specialized for

crowded dynamic spaces and allow good generalization to new environments by abstractly modeling environmental states; 2) better predict future states of the environment (*i.e.,* moving robots, moving pedestrians, and static obstacles) by taking into account the uncertainties of the environment; 3) design novel crowd-aware navigation control policies that can balance safety and speed and have strong generalizability by designing reward functions and utilizing abstract environment representations; 4) design reliable uncertainty-aware navigation control policies that are robust to the uncertainties in environment prediction and control actions by estimating and incorporating the uncertainties in observations and actions into control policies.

In summary, the overall goal of this research dissertation is to develop hardware-friendly learning-based navigation systems with strong generalizability to enable autonomous navigation in crowded dynamic scenes filled with static obstacles, moving robots, and dense pedestrians, providing highly innovative solutions to the open problems of autonomous navigation in crowded dynamic scenes. The overall dissertation outline is shown in Fig. 1.3. Chapter 1 introduces the research background of this dissertation and the research problems to be solved. Chapter 2 provides the necessary foundational knowledge on deep learning techniques used in this dissertation. Chapter 3 provides a detailed literature review of the mobile robot navigation problem, including environmental data processing and representation and control policy design. Chapter 4 presents a new set of preprocessed intermediate environmental data representations that are specialized for navigating crowded dynamic spaces and allow good generalization to new environments, which was originally published in [6]. Chapter 5 presents a novel learning-based crowd-aware control policy with strong generalizability by designing a novel velocity obstacle-based reward function that

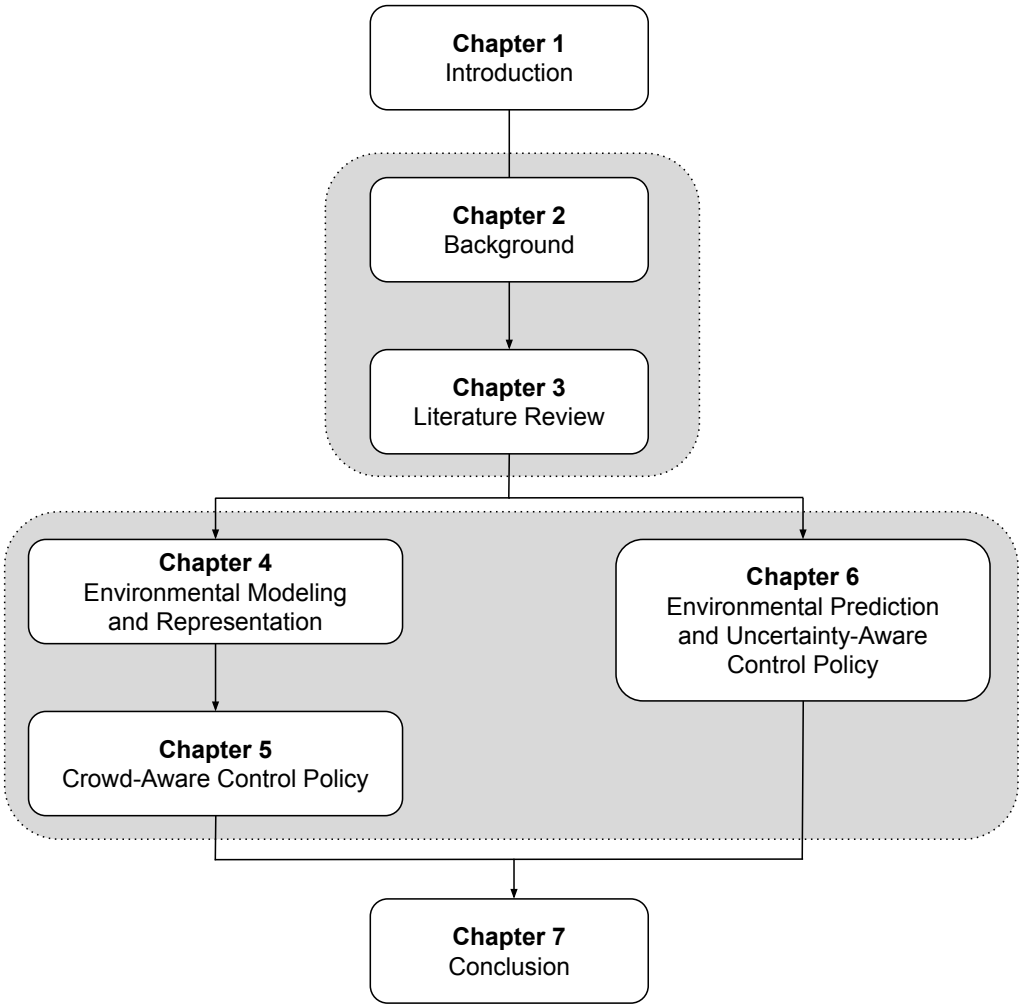



Figure 1.3. Dissertation outline.

can balance safety and speed, which was originally published in [7, 8]. Chapter 6 presents a family of novel stochastic environmental predictors for resource-constrained robots to predict a distribution of possible future states in dynamic scenes, as well as a novel predictive uncertainty-aware navigation framework to improve current existing model-based and learning-based control policies, which was originally published in [9, 10]. Finally, Chapter 7 summarizes the contributions of this paper and proposes open issues to be addressed in future work.

# CHAPTER 2
# BACKGROUND

This chapter begins with a brief introduction to the fundamental concepts and techniques that will be used throughout the dissertation. Specifically, we first briefly introduce the basic concepts of deep learning and different types of network architectures. We then provide a general tutorial on three different categories of learning techniques: supervised learning, unsupervised learning, and reinforcement learning.

## 2.1 Deep Learning Introduction

As shown in Fig. 2.1, Deep learning technology is a subset of machine learning technology that uses multi-layered neural networks (*i.e.,* no less than 3 layers) to simulate the complex decision-making capabilities of the human brain. Deep learning is the representation learning method that requires large amounts of data and powerful computing power to learn useful information and provide decision-making capabilities. Therefore, with the launch of ImageNet [11], the first large-scale image dataset, and the increase in GPU computing speed, deep learning technology became popular in the field of computer vision around the 2010s, and was quickly used in other diverse application areas such as natural language processing, healthcare, and robotics over the next decade.

Since data is a key component of deep learning, researchers developed three common deep neural network architectures to handle three typical data categories: fully connected neural network (FCNN) to handle vector-like data such as lidar sensor data or any tabular data, convolutional neural network (CNN) to handle grid-like data such as images or any data with spatial relationships, and recurrent neural network (RNN) to handle sequential data such as text data or any sequential data. In order to build these three common network

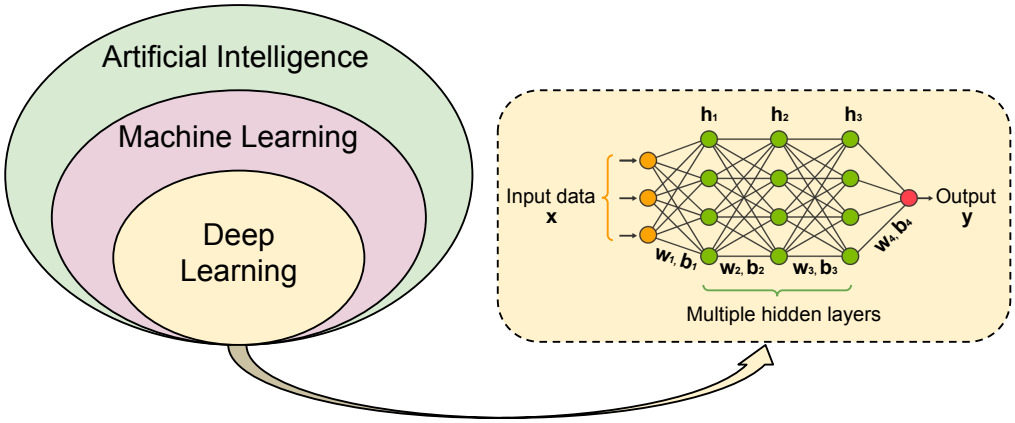



Figure 2.1. An illustration of the deep learning structure.

architectures, we need to introduce a key layer, which guarantees the strong approximation performance of deep neural networks: the activation layer. The activation layer is a crucial component layer in a neural network that uses activation functions to add nonlinearity to the network. While other types of neural network layers only provide linear operations or matrix multiplications, this nonlinearity is essential and allows neural networks to learn complex data patterns and model any nonlinear relationships in real-world data. The common activation functions include sigmoid, rectified linear unit(ReLU), and hyperbolic tangent (Tanh), which are shown in Fig. 2.2.

In simple terms, deep neural networks are composed of several linear operation layers and nonlinear operation layers. If we use the three-layer deep neural network model in Fig. 2.1 as an illustration example, the input data is represented by $\mathbf{x}$, the desired output data is represented by $\mathbf{y}$, the output of hidden layers is represented by $\mathbf{h}$, the deep neural network model is represented by $f$, the model parameters are represented by the weights $\mathbf{w}$

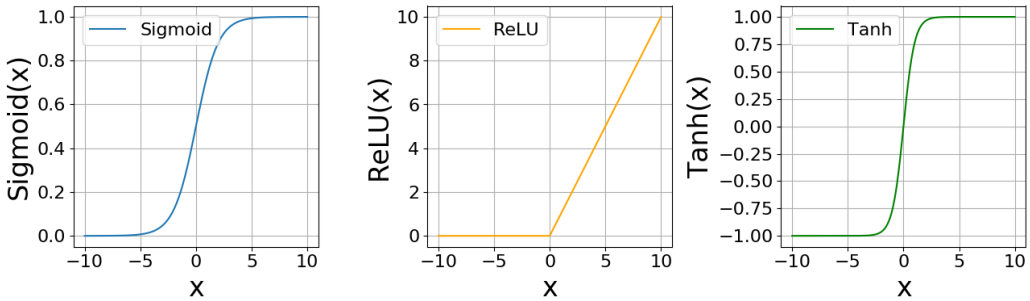



Figure 2.2. An illustration of different activation functions.

and the biases $b$, and the nonlinear operation is represented by $\sigma(\cdot)$. Then, our three-layer deep neural network model $\mathbf{y} = f(\mathbf{x}, \mathbf{w}, b)$ can be expressed mathematically at each layer:

$$\mathbf{h_1} = \sigma(\mathbf{x} \cdot \mathbf{w_1} + b_1), \tag{2.1a}$$

$$\mathbf{h_2} = \sigma(\mathbf{h_1} \cdot \mathbf{w_2} + b_2), \tag{2.1b}$$

$$\mathbf{h_3} = \sigma(\mathbf{h_2} \cdot \mathbf{w_3} + b_3), \tag{2.1c}$$

$$\mathbf{y} = \sigma(\mathbf{h_3} \cdot \mathbf{w_4} + b_4). \tag{2.1d}$$

All three common deep network architectures can be summarized mathematically as (2.1). The key difference lies in the choice of linear operation layers (*e.g.,* fully connected layer, convolutional layer, and recurrent layer).

### *2.1.1 Fully Connected Neural Network*

A fully connected neural network is a deep neural network in which every neuron in one hidden layer is connected to every neuron in the next hidden layer. It has dense connections, and its information transfer direction is feedforward without any loops. The key linear operation layer of FCNNs is called the fully connected layer, which applies a linear transformation to the input vector through many weight vectors. Therefore the key

components of FCNNs are fully connected layers and activation layers. As mentioned in Sec. 2.1, FCNNs are used to handle vector-like data to solve 1D data classification, regression, and function approximation tasks. The most popular FCNN is the multilayer perceptron (MLP) [12], which has also become the basic module for other more complex neural network architectures. However, FCNNs can be computationally expensive and prone to overfitting in high-dimensional data due to dense connections of each neuron.

### *2.1.2 Convolutional Neural Network*

In order to solve the problem of high computational complexity of FCNNs and process grid-like high-dimensional data, researchers proposed the convolutional neural network architecture. The key linear operation layer of CNNs is called the convolutional layer, which applies a linear transformation to the input matrix through many small convolution kernels. Due to the kernel convolution operation, it has relatively sparse connections, and its information transmission direction is also feedforward without any loops. In addition, to further reduce the computational load, convolutional layers are often combined with pooling layers (*e.g.,* average pooling and maximum pooling). Pooling layers are used to reduce the spatial dimensions of the grid-like feature maps to reduce computational load and control overfitting in high-dimensional data. Therefore the key components of CNNs are convolutional layers, activation layers, pooling layers, and fully connected layers. Note that CNNs are well suited for processing grid-like visual data and solving computer vision-related tasks, including image classification, object detection, and image segmentation. There are many famous and typical CNN-based networks, such as LeNet [13], AlexNet [14], VGGNet [15], and ResNet [16], used to solve these computer vision-related

tasks. However, like FCNNs, CNNs also use a feedforward network architecture and cannot handle sequential data, limiting their application in time series forecasting and analysis.

### *2.1.3 Recurrent Neural Network*

In order to handle temporal and sequential data, researchers proposed the recurrent neural network architecture. Unlike FCNNs and CNNs, which use feedforward network architectures, RNNs use a feedback network architecture, which has connections that form directed loops and allows them to maintain a "memory" of previous inputs by using their internal state. This feedback connection is implemented by the key linear operation layer (recurrent layer) of RNNs, where the recurrent layer has a hidden state to store previous information and each neuron in the recurrent layer can receive input from previous steps in the sequence, allowing information to be retained. The key components of RNNs are recurrent layers, activation layers, and fully connected layers. Due to their "memory" properties, RNNs are particularly effective for solving sequential data tasks, such as time series prediction, natural language processing, and video analysis. There are two main and popular variants of the Vanilla RNN used in current real-world applications: the long short-term memory (LSTM) [17] and gated recurrent unit (GRU) [18]. LSTM aims to overcome the vanishing gradient problem of Vanilla RNN, while GRU aims to simplify the complex structure of LSTM. However, RNNs have a higher computational load compared to CNNs and FCNNs because they need to maintain memory of previous inputs.

## 2.2 Primary Learning Techniques

In addition to designing deep neural network architectures, another key issue is how to train these deep neural networks with data. We can regard this deep neural network

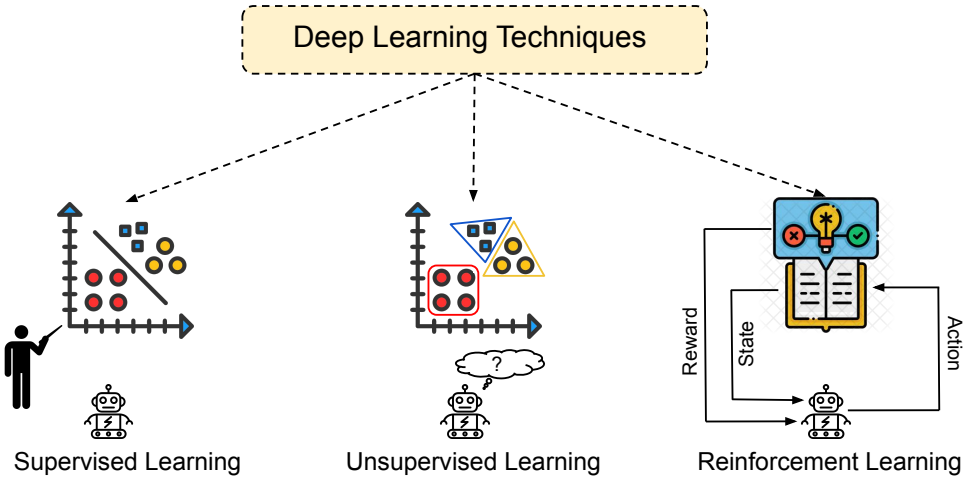



Figure 2.3. Three primary deep learning techniques.

training problem as an optimization problem. If we go back to our three-layer deep neural network example in Fig. 2.1 and use $\theta$ to denote the model parameters (*i.e.,* $\mathbf{w}$ and $\mathbf{b}$), then our deep neural network model can be expressed as:

$$\mathbf{y} = f_\theta(\mathbf{x}). \tag{2.2}$$

Then, the objective of our training problem is to find the optimal model parameters $\theta^*$ to minimize the loss $L$ between the output $\mathbf{y}$ and the desired output data $\mathbf{y}^*$, which can be expressed as:

$$\theta^* = \arg\min_\theta \mathrm{L}_\theta(\mathbf{y}, \mathbf{y}^*). \tag{2.3}$$

Three primary learning techniques used to solve this optimization problem and train deep neural network models are supervised learning, unsupervised learning, and reinforcement learning, as shown in Fig. 2.3.

### *2.2.1 Supervised Learning*

Supervised learning is a type of deep learning that uses labeled datasets to train deep neural networks to learn relationships between inputs and outputs. The desired output data $\mathbf{y}^*$ in (2.3) is the ground truth label from the labeled dataset. The key idea of supervised learning is creating a well-labeled dataset, where the demonstrator or teacher provides accurate required label data to teach the student (*i.e.,* deep neural network) to learn the underlying patterns between network inputs and outputs. However, dataset collection and labeling are time-consuming and labor-intensive. Supervised learning is the most popular learning technique for training deep neural networks using labeled datasets. It is well suited for solving tasks in three main areas: classification, regression, and object detection.

The common supervised learning loss functions $L$ in (2.3) include classification losses (*e.g.,* cross-entropy loss and hinge loss) and regression losses (*e.g.,* mean squared error (MSE) loss and mean absolute error (MAE) loss). With a supervised learning loss function defined, a common algorithm for solving the optimization problem in (2.3) is the stochastic gradient descent (SGD) algorithm, as shown in Algorithm 2.1. Note that we use the supervised learning technique to solve the object detection and navigation control tasks in our mobile robot navigation problems.

### *2.2.2 Unupervised Learning*

Unlike supervised learning, where the deep neural network model learns from the labeled dataset, unsupervised learning is a type of deep learning that uses only unlabeled input data to train deep neural networks to self-learn relationships behind unlabeled input data. There is no labeled ground truth data or the desired output data $\mathbf{y}^*$ in (2.3) is the

**Algorithm 2.1:** Stochastic Gradient Descent

**Input:** training dataset $\mathbf{D} = (\mathbf{x}, \mathbf{y}^*)$

**Input:** the initial model parameters $\theta_\mathbf{0}$

**Input:** learning rate or step size $\alpha$

**Input:** batch size $m$

**Input:** the number of training steps $N$

**Output:** the optimal model parameters $\theta^*$

1: **for** k $= 1, 2, 3, \ldots, N$ **do**

2: Extract $m$ dataset sample pairs: $\mathbf{D_i} = (\mathbf{x_i}, \mathbf{y_i^*}) \leftarrow$ sample from $\mathbf{D} = (\mathbf{x}, \mathbf{y}^*)$

3: Do deep neural network model inference: $\mathbf{y_i} = f_\theta(\mathbf{x_i})$

4: Update model parameters: $\theta_{\mathbf{k+1}} = \theta_\mathbf{k} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla L_\theta(\mathbf{y_i}, \mathbf{y_i^*})$

5: **end for**

6: **return** $\theta^* = \theta_\mathbf{N}$

input data $\mathbf{x}$ itself. The key idea of unsupervised learning is using a deep neural network to self-learn the underlying patterns or structures within the unlabeled data on its own. It is designed to solve problems that researchers do not know the answers to, especially when they have large amounts of data. Therefore unsupervised learning is well suited for solving tasks in three main areas: clustering, association, and reconstruction.

The common unsupervised learning loss functions $L$ in (2.3) include reconstruction losses ((*e.g.,* mean squared error (MSE) loss and binary cross-entropy (BCE) loss)) and clustering losses (*e.g.,* Kullback-Leibler (KL) divergence loss). Unsupervised learning also uses the stochastic gradient descent algorithm in Algorithm 2.1 to train deep neural network models. Note that we the unsupervised learning technique to solve the environment reconstruction and prediction tasks in our mobile robot navigation problems.

### *2.2.3 Reinforcement Learning*

Unlike supervised learning and unsupervised learning, which use datasets for training, reinforcement learning is a type of deep learning that uses the experience of interaction with the environment to train deep neural network-based agents to learn to make decisions. It involves learning through trial and error and adjusting an agent's behavior in the environment based on feedback in the form of rewards or punishments. The key idea of reinforcement learning is to maximize the cumulative reward of a deep neural network-based agent acting in the environment over time. Therefore reinforcement learning is well suited for solving sequential decision-making tasks.

Although there is no desired output data to guide the optimization of reinforcement learning, there is customized reward feedback. In order to use reinforcement learning techniques to train deep neural network-based agents, we need to rewrite the loss function $L$ in (2.3) in terms of cumulative rewards instead of desired output data $\mathbf{y}^*$. To do this, we first formulate the sequential decision-making process as a partially observable Markov decision process (POMDP), which is denoted by a 6-tuple $\langle S, A, T, R, \Omega, O \rangle$ where $S$ is the state space, $A$ is the action space, $T$ is the state-transition model, $R$ is the reward function, $\Omega$ is the observation space, and $O$ is the observation probability distribution. Then, the general objective $L(\theta)$ of reinforcement learning training is to maximize the expected discounted return by optimizing the deep neural network model:

$$L(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R^t\right], \tag{2.4}$$

where $\gamma$ is the discount rate in the range of $[0,1]$, $\mathbb{E}_{\pi_\theta}[\cdot]$ denotes the expected value of a random variable assuming the agent follows the deep neural network model $f_\theta(\mathbf{x})$.

There are two main types of deep reinforcement learning algorithms to help train deep neural network-based agents: model-based algorithms and model-free algorithms. The key difference between model-based and model-free algorithms is whether they require the use of a model of the environment that predicts future states and rewards. Model-based algorithms require such an environment model, while model-free algorithms do not. The typical model-based algorithms include world models [19], imagination-augmented agents (I2A) [20], model-based reinforcement learning with model-free fine-tuning (MBMF) [21], and model-based value expansion (MBVE) [22]. The typical model-free algorithms include deep Q-networks (DQN) [23], deep deterministic policy gradient (DDPG) [24], twin delayed DDPG (TD3) [25], asynchronous advantage actor-critic (A2C/A3C) [26], trust region policy optimization (TRPO) [27], and proximal policy optimization (PPO) [28]. Due to the difficulty of obtaining/learning an environment model, most robotics researchers choose model-free algorithms to train their robotic agents, especially using the PPO algorithm (as shown in Algorithm 2.2). A major advantage of the PPO algorithm is that it is much simpler to implement and can achieve comparable or better results than other state-of-the-art algorithms (*e.g.,* A2C [26], TRPO [27] and ACER [29]). Note that we use the reinforcement learning technique with the PPO algorithm to solve the navigation control tasks in our mobile robot navigation problems.

**Algorithm 2.2:** PPO-Clip

**Input:** intial policy parameters $\theta_{\mathbf{0}}$

**Input:** initial value function parameters $\phi_{\mathbf{0}}$

**Input:** the number of training steps $N$

**Output:** the optimal policy parameters $\theta^*$

**Output:** the optimal value function parameters $\phi^*$

1: **for** k $= 1, 2, 3, \ldots, N$ **do**

2: Collect set of trajectories: $\mathbf{D_k} = \{\tau_i\} \leftarrow$ running $f_{\theta_\mathbf{k}}(\mathbf{x_k})$ in the environment

3: Compute the reward estimates: $\hat{R}_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

4: Compute the advantage estimates: $\hat{A}_t = \hat{R}_t - V_\phi(s_t)$

5: Update the policy parameters by maximizing the PPO-Clip objective:
$\theta_{\mathbf{k+1}} = -\arg\min_{\theta} \frac{1}{|\mathbf{D_k}|T} \sum_{\tau \in \mathbf{D_k}} \sum_{t=0}^{T} \min \left[ \frac{f_{\theta_{\mathbf{k+1}}}(a_t|s_t)}{f_{\theta_\mathbf{k}}(a_t|s_t)} A^{f_{\theta_\mathbf{k}}}(s_t, a_t), g(\varepsilon, A^{f_{\theta_\mathbf{k}}}(s_t, a_t)) \right]$
typically via Algorithm 2.1

6: Fit value function by regression on mean squared error:
$\phi_{\mathbf{k+1}} = \arg\min_{\phi} \frac{1}{|\mathbf{D_k}|T} \sum_{\tau \in \mathbf{D_k}} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2$
typically via Algorithm 2.1

7: **end for**

8: **return** $\theta^* = \theta_\mathbf{N}$, $\phi^* = \phi_\mathbf{N}$

## 2.3 Deep Neural Network Optimization

Due to limited computing capability, energy, memory, and storage, resource-constrained robots always need hardware-friendly deep learning algorithms. Before designing a deep learning algorithm for a robot, it is important to consider the computational resources in the robot and maintain a good balance between performance and computational efficiency. Many researchers have been working on accelerating deep learning algorithms from hardware and software aspects, or both. Compared to hardware acceleration, which requires significant redesign and manufacturing changes using specialized hardware architectures

such as field programmable gate arrays (FPGAs), neural processing units (NPUs), and custom application-specific integrated circuits (ASICs) (most robots are not equipped with these specialized hardware computing devices), software acceleration offers greater advantages by enabling flexibility and adaptability through modifications to deep neural networks and other software optimizations. As summarized in surveys [30, 31], the mainstream software acceleration methods can be divided into four categories: network pruning, weight quantization and sharing, knowledge distillation, and customized network design.

### *2.3.1 Network Pruning*

The key idea of network pruning is to analyze the different components of complex deep neural networks and remove unimportant components, such as unimportant channels [32, 33], unimportant kernel filters [34, 35], unimportant network connections [36, 37], and even unimportant layers [38, 39]. However, network pruning requires expertise and experience to analyze the network model and can easily lead to performance degradation.

### *2.3.2 Weight Quantization and Sharing*

To avoid the extra knowledge of pruning, many researchers focus on network model weight quantization and sharing, such as using Huffman coding to quantize weights [40], reduce the number of bits representing model weights [41, 42], and share weights between different network connections or different layers [43, 44]. However, weight quantization often requires a good trade-off between performance and bit quantization, and requires hardware computing devices to support its low-precision arithmetic operations.

### *2.3.3 Knowledge Distillation*

To keep network model performance and compress neural network, Hinton et al. [45] first proposed the knowledge distillation technique, the key idea of which is to train a small and simple "student" network model to learn the output of a large and complex "teacher" network model. Following the basic distillation idea, many improvement variants are proposed to reduce the learning gap between the "student" network and the "teacher" network such as adding a middle-size network as the "teaching assistant" [46, 47], using early-stopped knowledge distillation "teacher" network [48], and incorporating cross-domain distillation training [49]. Although knowledge distillation requires the selection of appropriate "student" or "teacher" networks, it provides more freedom to choose different compressed network architectures and provides the potential possibility of obtaining compressed "student" network models with better generalization capabilities. Note that we use knowledge distillation techniques to software optimize deep neural networks for the task of environment prediction in our mobile robot navigation problems.

### *2.3.4 Customized Network Design*

A more liberal option for compressing deep neural networks is to directly custom design an efficient small neural network to implement the desired task. For example, Iandola et al. [50] propose an efficient SqueezeNet network architecture by using 1x1 convolutions and delayed downsampling to maintain a small model size without compromising accuracy. Google researchers propose the efficient MobileNet series (*i.e.,* MobileNetV1 [33], MobileNetV2 [51], and MobileNetV3 [52]) that uses depthwise convolutions and point-wise convolutions to replace the standard convolution to build lightweight deep neural

networks. Similarly, Megvii researchers propose the efficient ShuffleNet series (*i.e.,* ShuffleNetV1 [53] and ShuffleNetV2 [54]) by using group convolutions and channel shuffle operations to reduce the number of computations and parameters. All of these efficient network architectures have been widely used in different computer vision applications, such as image classification, object detection, and semantic segmentation. Although designing such an efficient network architecture requires some specific knowledge and wizardry, it shows great potential in balancing the trade-off between accuracy and computational efficiency and is suitable for robotics applications. Note that we also customize hardware-friendly deep neural networks based on our data representation to provide navigation control policies in our mobile robot navigation problems.

## 2.4 Summary

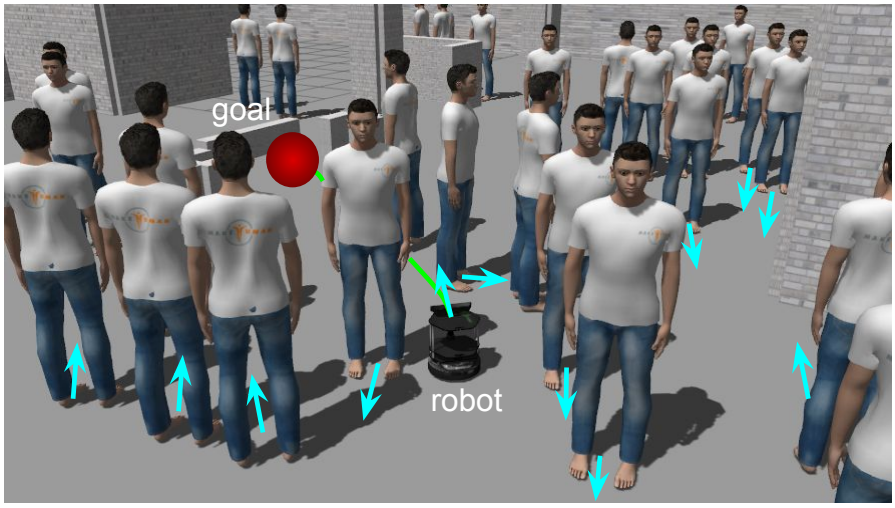
In this chapter, we briefly introduced the basics of deep learning technology, including basic neural network architectures, training methods, and neural network compression and optimization techniques. These materials are the cornerstone for all the work in this dissertation. We use them to help design and implement all learning-based algorithms for open problems in mobile robot navigation, such as environment perception, representation, prediction, and control policy design.

# CHAPTER 3
# LITERATURE REVIEW

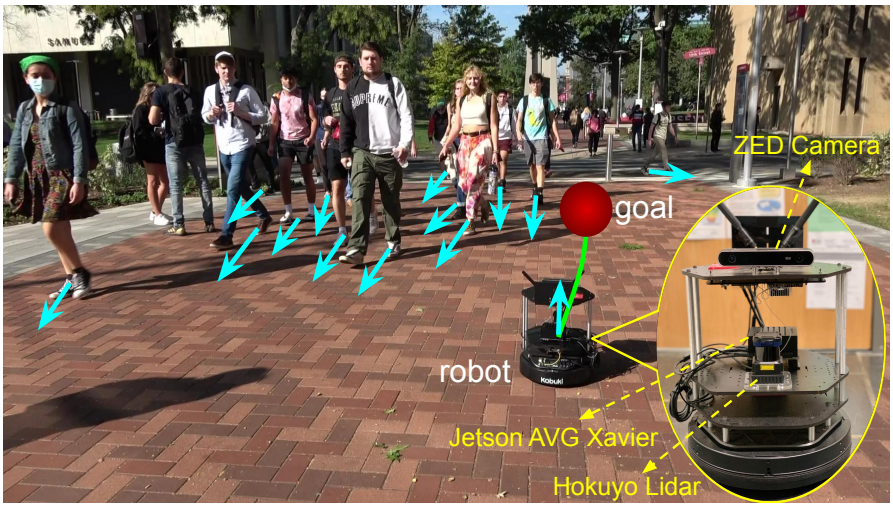
## 3.1 Overview

As discussed in Chapter 1, one common application of autonomous mobile robots is replacing manual labor to provide last-mile delivery services. All these delivery tasks have time limits and require mobile robots to navigate autonomously and quickly to destinations through a partially known space filled with moving people and other static obstacles, as shown in Fig. 3.1. The main challenges faced by these mobile robots are perceiving complex environments, especially unknown and dynamic pedestrians; extracting useful information; and generating a policy that yields autonomous navigation.

Beginning in the 1980s, researchers began to construct occupancy grid maps of the surrounding environment for robots and used path planning algorithms (*e.g.,* Dijkstra's [55], A* [56], and Rapidly-Exploring Random Trees (RRT) [57]) as global planners to solve mobile robot navigation problems. The basic framework of this initial navigation stack is shown in Fig. 3.2(a). To successfully navigate to the destination, the global planner provides a collision-free path for a mobile robot from the current position to the goal position by acquiring a full/partial map of the environment, the goal position, the robot position, and even surrounding obstacles. The robot then simply sends control commands to its actuators to follow this collision-free path to its destination. If any new obstacles are detected and the cost map is changed, the global planner will re-plan a new global path for the robot. Although such a navigation stack with a single global planner is straightforward, it is computationally inefficient if the environment is dynamic and changing. Furthermore,




(a) Simulated indoor environment




(b) Real-world outdoor environment

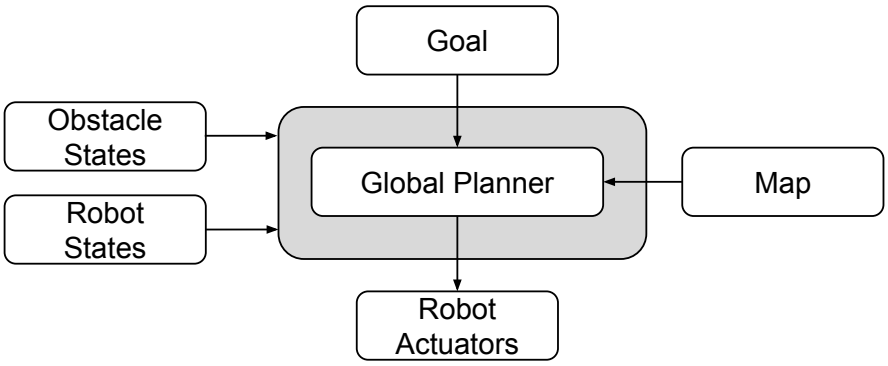
Figure 3.1. Illustration of the robot navigation problem. The robot navigates through crowds in a simulated indoor environment and a real-world outdoor environment. The robot is following a nominal path (green line) to the goal (red disk) while avoiding pedestrians (blue arrows show current velocity).

even state-of-the-art planning algorithms cannot effectively deal with moving obstacles and consistently provide collision-free paths in dynamic environments [58].

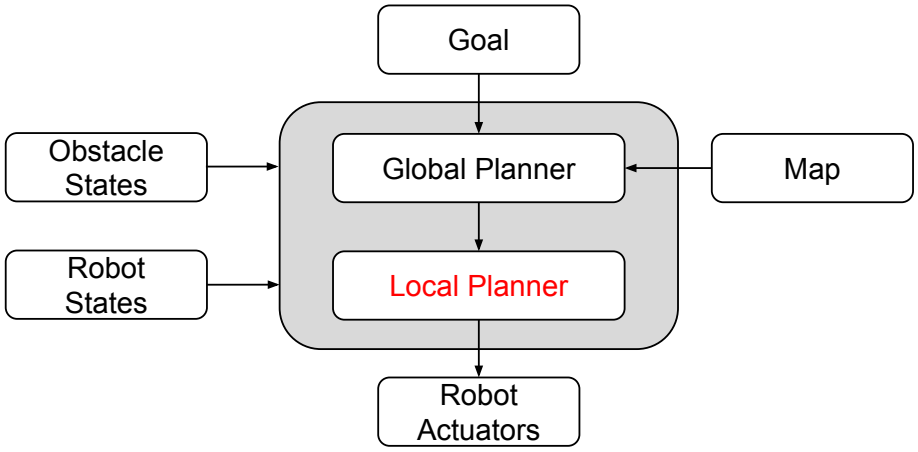
To reduce computational stress and improve collision avoidance performance for moving obstacles, a reactive local planner is added to the initial navigation stack to handle moving obstacles and local changes in the environments. This new framework combining

global planner and local planner has become the mainstream framework of robot navigation, as shown in Fig. 3.2(b). In this new framework, the global planner only needs to provide the nominal path to the final goal position according to the known map, and the local planner uses the perceived obstacle states in the local perception window and the kinematics and dynamics of the robot to avoid moving obstacles in real time. The most typical local planners are built by dynamic window approach (DWA) [5] and velocity obstacle (VO)-based approaches [59–61] proposed from the 1990s. In particular, the DWA local planner, which is incorporated into the Robot Operating System (ROS) [62] navigation stack with a global planner (using the framework with both global and local planners), is widely used in current commercial mobile robots. However, these navigation stacks with different model-based local planners (*e.g.,* DWA and VOs) only work well in sparse dynamic environments and are still unable to handle crowded dynamic environments.

To achieve autonomous navigation in crowded dynamic scenes, under the navigation framework of global and local planners, researchers are currently trying to design local collision avoidance planners using deep learning techniques such as supervised learning, unsupervised learning, and reinforcement learning [63]. The basic structure of learning-based control approaches is shown in Fig. 3.3, which can be divided into the forward inference stage and the backward training stage. In the forward inference stage, input data, which can be raw sensor data (*e.g.,* lidar scans and camera images) or preprocessed sensor data (*e.g.,* pedestrian current motions and trajectory predictions), and the goal position are directly fed into the control policy to calculate the desired control velocities online. Note that, unlike traditional model-based control policies that require multi-stage formula computation or optimization, the learning-based control policy is simply represented by a deep




(a) Framework with a single global planner




(b) Framework with both global and local planners

Figure 3.2. Basic frameworks of the robot navigation stack.

neural network (DNN). A deep neural network is a black box for approximating complex computational processes. The typical DNN architectures include fully connected neural networks (FCN), convolutional neural networks (CNN), recurrent neural networks (RNN), graph neural networks (GNN), and transformer neural networks. The backward training stage is used to train the control policy network offline. The first step in the backward training stage is to feed the output control velocities into the loss calculation module to calculate the loss. While supervised learning-based approaches directly compare the difference between output and ground truth labels, and usually use cross-entropy and mean

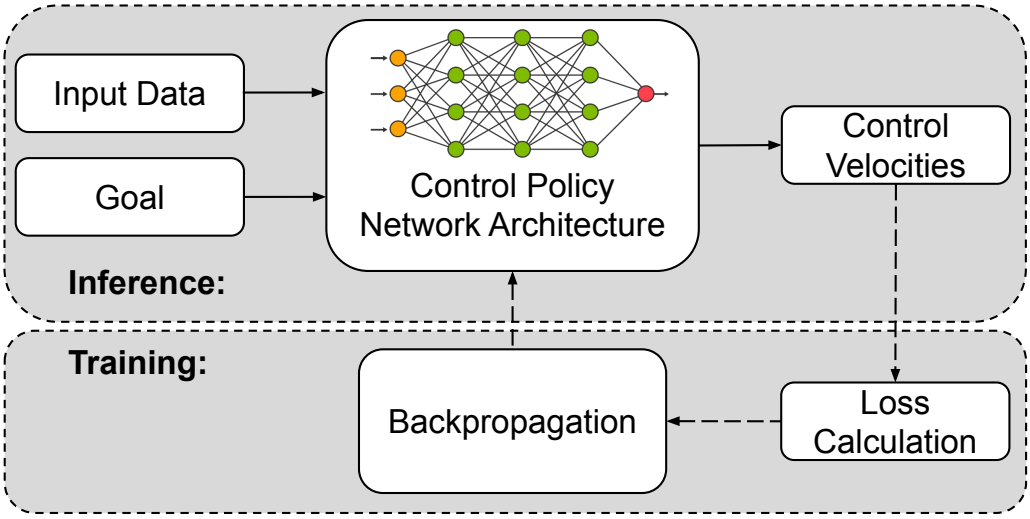



Figure 3.3. Basic structure of learning-based control approaches.

square error loss functions to calculate the loss, reinforcement learning-based approaches indirectly use experienced reward functions to evaluate the goodness of the output and use different types of algorithms such as deep deterministic policy gradient (DDPG) [24] and proximal policy optimization (PPO) [28] to calculate the loss. After calculating the loss, the backpropagation module computes the gradients of the loss function with respect to each neural network layer and backpropagates them layer by layer to the policy network. Finally, gradient descent methods such as stochastic gradient descent (SGD) [64] and Adam [65] can be used to train the policy network by automatically updating network parameters to minimize loss.

In this chapter, we describe in detail previous work on the robot navigation problem from the 1980s to the present, especially the processing of environmental data and the design of local planners. The processing of environmental data includes environmental data perception and representation, environmental data collection, and environmental data prediction. The design of local planners can be divided into two broad categories: model-

based and learning-based. Among them, learning-based approaches can be further divided into supervised learning-based and reinforcement learning-based.

## 3.2 Environmental Data Processing

### *3.2.1 Environmental Perception and Representation*

The first prerequisite for robots to navigate and perform tasks is to use their sensors to perceive the surrounding environment. The main sensors for robot perception can be divided into two categories: range sensors and vision sensors. Range sensors include ultrasonic sensors, infrared sensors, lidar sensors, and radar sensors. Vision sensors include many different types of camera sensors such as grey cameras, RGB cameras, depth cameras, and stereo cameras. Due to technical limitations in the 1990s, ultrasonic sensors and infrared sensors were widely used for robot navigation. For example, Fox et al. [5] use ultrasonic sensors to perceive the environment and create obstacle costmaps to represent complex environments. Flynn et al. [66] combine ultrasonic sensors with infrared sensors to sense the environment and build the curvature primal sketch representation to represent the detected objects. However, these two types of range sensors have low accuracy and short detection range.

As technology develops, researchers are using lidar sensors with higher accuracy and longer detection ranges to provide environmental perception. Ferrer et al. [67] use lidar information to detect people and other static objects and provide obstacle information representation for their social force navigation model. Many learning-based navigation works [68–73] directly use raw lidar scan data directly as input data representation and feed it into their neural networks to provide navigation behavior. However, these distance

sensors are more suitable for providing information about geometric shapes and their distances, but have difficulty in providing information about object categories.

To perceive more meaningful semantic object information, some learning-based navigation works [74–76] similarly directly use raw camera RGB images as input data representation and feed them into their neural networks to implicitly extract useful object information. Instead of using raw camera data representation, Liu et al. [77] use object detection algorithms to preprocess the raw camera RGB and depth data to obtain pedestrian position and velocity information as the data representation of their control policy network. Furthermore, many other learning-based navigation works [78–80] preprocess both camera data and lidar point cloud data to extract the position and velocity of pedestrians as the data representation. Similarly, other works [81–83] have combined lidar sensors and camera sensors to capture and model complex robot-human interactions, which helps robot navigation. Sathyamoorthy et al. [84] and Huang et al. [85] also leverage preprocessed data from lidar and camera sensors to predict pedestrian trajectories and used them as preprocessed data representation to improve safe robot navigation.

Following this trend, we can see that environmental perception and data representation are important issues for robot navigation because they determine the robot's ability to perceive its surroundings. How to fuse different types of sensors and design a suitable preprocessed data representation is still an open problem to be solved.

### *3.2.2 Environmental Prediction*

After mobile robots use their sensors to perceive the state of their environments, another important option is to accurately and reliably predict how the surrounding environ-

ment will change based on these sensor data. This will allow mobile robots to proactively act to avoid potential future collisions, a key part of autonomous robot navigation.

Environment prediction remains an open problem as the future state of the environment is unknown, complex, and stochastic. Many interesting works have focused on this prediction problem. Traditional object detection and tracking methods [86, 87] use multi-stage procedures, hand-designed features, and explicitly detect and track objects. More recently, deep learning (DL)-based methods that are detection and tracking-free have been able to obtain more accurate predictions [88–93]. Occupancy grid maps (OGMs), the most widely successful spatial representations in robotics, are the most common environment representations in these DL-based methods. This transforms the complex environment prediction problem into an OGM prediction problem, outlined in Fig. 6.1. Since OGMs can be treated as images (both are 2D arrays of data), the multi-step OGM prediction problem can be thought of as a video prediction task, a well-studied problem in machine learning.

The most common technique for OGM prediction uses recurrent neural networks (RNNs), which are widely used in video prediction (*e.g.,* ConvLSTM [94], PredNet [95], and PhyDNet [96]). For example, Ondruska et al. [88] first propose an RNN-based deep tracking framework to directly track and predict unoccluded OGM states from raw sensor data. Itkina et al. [89] directly adapt PredNet [95] to predict the dynamic OGMs (DOGMas) in urban scenes. Following this line of thought, Toyungyernsub et al. [90] decouple the static and dynamic OGMs and propose a double-prong PredNet to predict occupancy states of the environment. Similarly, Schreiber et al. [91, 92] embed the ConvLSTM units in the U-Net architecture to capture spatio-temporal information of DOGMs and predict them in the stationary vehicle setting. Lange et al. [93] propose two attention-augmented

ConvLSTM networks to capture long-range dependencies and predict future OGMs in the moving vehicle setting. However, these image-based works only focus on improving network architectures and just treat the OGMs as images, assuming their network architectures can implicitly capture useful information from the kinematics and dynamics behind the environment with sufficient good data.

While pure image-based methods are simple and explicitly ignore the dynamic information of the environment, many other DL-based approaches exploit the ego-motion information and motion flow of the environment to improve the OGM prediction accuracy. By using a combination of input placement and recurrent states shifting to compensate for the ego-motion, Schreiber et al. [97] extend their previous image-based works [91, 92] to predict DOGMs in moving ego-vehicle scenarios. By extending the deep tracking framework [88] with a spatial transformer module to compensate for ego-motion, Dequaire et al. [98] propose a gated recurrent unit (GRU)-based deep tracking network with a spatial transformer module (STM) for ego-motion compensation to predict multi-steps future states in stationary and moving vehicle settings. Song et al. [99] propose a GRU-based LiDAR-FlowNet to estimate the forward and backward motion flow between two consecutive OGMs and predict future OGMs. Thomas et al. [100, 101] directly encode spatiotemporal information into the world coordinate frame and propose a 3D-2D feedforward architecture, called DeepSOGM, to predict futures. By considering the ego-motion and motion flow together, Mohajerin et al. [102] first use the standard geometric image transformation to compensate for the ego-motion of the vehicle, then propose a ConvLSTM-based difference learning architecture to extract the motion difference between consecutive OGMs, and finally predict multi-step future states. However, most of these motion-based works are de-

signed for vehicle scenarios, and their network models have a large memory footprint and require significant computation, putting them out of the range for small mobile robots with limited resources. Furthermore, all the above-described works only provide deterministic OGM predictions and cannot estimate the uncertainty of future states, which is a key point in helping robots operate in complex, dynamic, and hazardous environments. How to provide effective stochastic environmental prediction with uncertainty estimation in complex dynamic environments remains an important problem to be solved.

## 3.3 Control Policy Design

### *3.3.1 Model-based Approaches*

A typical model-based approach is the ROS navigation stack, which uses costmaps to store obstacle information and uses the dynamic window approach (DWA) planner [5] to do local planning. Based on the ROS navigation stack, Singamaneni et al. [103] recently add human safety and visibility costmaps into both global and local planners to handle human-aware navigation problems. Another class of model-based approaches are velocity obstacle-based algorithms [59–61], which map the dynamic environment into the robot velocity space and generate safe control velocities from these velocity constraints. Similarly, Arul et al. [104] combine reciprocal velocity obstacles with buffered Voronoi cells to solve multi-agent navigation problems. A third class of model-based approaches uses model predictive control [105, 106], which can integrate collision avoidance and obstacle dynamics into robot constraints. Other model-based approaches attempt to directly model how humans and robots interact, using social forces [67, 107], Gaussian mixture models (GMM) [108], or a combination of potential functions and limit cycles [109]. All of those

model-based approaches require knowledge of pedestrian kinematics, utilize multi-stage procedures to process sensor data, and often require practitioners to carefully hand-tune model parameters, making them difficult to implement and generalize to new scenarios.

### *3.3.2 Supervised Learning-based Approaches*

With the success of deep learning in the computer vision area, many researchers started to apply learning-based approaches to other problems, including robot navigation. Unlike the traditional model-based approaches with multi-stage procedures, several recent works use deep neural networks to learn end-to-end control policies that generate steering commands directly from raw sensor data. Pfeiffer et al. [68] create a data-driven end-to-end motion planner that uses a convolutional neural network (CNN) to generate a linear and angular velocity from raw lidar data. Tai et al. [74] also use a CNN to select from one of five discrete robot control commands using raw depth camera data as input. Loquercio et al. [75] use a similar approach with raw camera image data to train a control policy that enables a drone to drive safely in the streets. Similarly, Kahn et al. [76] also use the raw camera image data to train an end-to-end learning-based mobile robot navigation system that can navigate in real-world environments with geometrically distracting obstacles. However, most of these end-to-end approaches only focus on static environments. For dynamic environments, Pokle et al. [73] use multiple CNN networks to learn multimodal high-level features from raw sensor data. A CNN-based local planner fuses these features to generate velocity commands. However, their learned feature-based control policy was only tested in a relatively open simulated environment with no more than 3 moving pedestrians. Furthermore, these policies are trained via supervised learning meth-

ods, which requires collecting a lot of expert demonstrations to train the network and learn safe navigation behaviors.

### *3.3.3 Reinforcement Learning-based Approaches*

Compared with supervised learning-based methods, DRL-based approaches are more commonly used for navigation in dynamic environments. Using the proximal policy optimization (PPO) algorithm [28], Long et al. [69, 70] first propose a fully decentralized multi-robot collision avoidance framework using 2D raw lidar range data. Following this line of thought, Guldenring et al. [71] investigate the different state representations of 2D lidar data, and use the PPO training algorithm to train a human-aware navigation policy. Arpino et al. [72] present a multi-layout training regime that uses raw lidar data as its input to train an indoor navigation policy. Huang et al. [85] construct a multi-modal late fusion network to fuse raw lidar data and raw camera image data. Although these sensor-level approaches can quickly and easily generate control policies, they cannot distinguish between static or dynamic obstacles, nor can they utilize high-level information such as the positions and velocities of obstacles.

To address the limitations of using only raw sensor data, many other works leverage information about pedestrian motion to enable safe navigation in dynamic environments. Everett et al. [78–80] directly feed the relative position and velocity of pedestrians into the neural network, and generate socially aware collision avoidance policies for pedestrian-rich environments. Chen et al. [110] focus on jointly modeling human-robot and human-human interactions by a self-attention mechanism. Immediately afterwards, they also propose a relational graph learning network to infer robot-human interactions and predict their tra-

jectories [81]. Similarly, another work [111] encodes the crowd state with a graph convolutional network (GCN) and predicts human attention. Recently, Liu et al. [77] use a decentralized structural-recurrent neural network (RNN) to model the interaction between the robot and pedestrians. One major drawback of these agent-level approaches is that they are only suitable for solving crowded navigation problems in open spaces and ignore static obstacles and geometric constraints in the layout. To solve this issue, Liu et al. [83] expand the work of Chen et al. [110] by adding additional static obstacle maps, but they need two parallel network models to handle the presence/absence of pedestrians. Sathyamoorthy et al. [84] expand the PPO training policy of Long et al. [69] by using a late fusion network to combine raw lidar data with pedestrian trajectory predictions. Dugas et al. [112] use unsupervised learning architectures to predict and reconstruct future lidar latent states. However, most of the above-described works only focus on exploiting the sensor perception information and its data representations, but ignore the reward function design.

Towards this, Xu et al. [113] propose a knowledge distillation reward term to encourage the robot to learn expert behaviors. Patel et al. [114] propose a square warning region-based reward function to encourage the robot to navigate in the direction opposite to the heading direction of obstacles. However, the effect of knowledge distillation reward largely depends on the quality of the expert dataset, and the reward function based on the square warning region is too conservative, resulting in slow actions. Reward function design in reinforcement learning-based policies remains an open problem for robot navigation in crowded dynamic environments.

Table 3.1. Summary of the different categories of robot navigation methods.

| **Approaches** | Model-based | Supervised learning-based | Reinforcement learning-based |
|---|---|---|---|
| **Literatures** | [5, 59–61, 67, 103–109] | [68, 73–76] | [69–72, 77–81, 83–85, 110–114] |
| **Application environments** | Static environments<br>Sparse dynamic environments | Static environments<br>Sparse dynamic environments | Static environments<br>Sparse dynamic environments<br>Crowded dynamic environments |
| **Environmental representation** | Raw 2D lidar scans<br>Robot positions/velocities<br>Obstacle positions/velocities | Raw 2D/3D lidar scans<br>Raw RGB/gray camera images<br>Raw depth camera images<br>Raw Point cloud data | Raw 2D/3D lidar scans<br>Raw RGB/gray camera images<br>Raw depth camera images<br>Raw Point cloud data<br>Robot positions/velocities<br>Pedestrian positions/velocities<br>Pedestrian trajectory predictions<br>Lidar state reconstructions/predictions<br>Human-robot interactions |
| **Tuning parameters** | Hand tuning | Data-driven tuning | Experience-driven tuning |
| **Advantages** | Rule-based<br>Easy to explain<br>Explicitly define constraints | Data-based<br>Behavior cloning<br>No curse of dimensionality | Experience-based<br>Similar to human learning<br>No curse of dimensionality |
| **Disadvantages** | Require expert knowledge<br>Manually tuning parameters<br>Multi-stage procedures<br>Curse of dimensionality | Black box<br>Lack of explainability<br>No safety guarantees<br>Laboriously collect data | Black box<br>Lack of explainability<br>No safety guarantees<br>Sim-to-real gap<br>Carefully design reward function |

## 3.4 Comparative Analysis

There is an abundance of studies that focus on robot navigation problems, with solutions ranging from traditional model-based approaches to learning-based approaches (*i.e.,* supervised learning-based approaches and reinforcement learning-based approaches). Table 3.1 summarizes the general characteristics of these three different categories of methods from different features such as environmental data representation and their advantages/disadvantages.

Typical model-based approaches compute efficient paths and the parameters are easily interpretable, but require manually adjusting model parameters for different scenarios,

making them difficult to implement and adapt to new settings [5, 59–61, 67, 103–109]. On the other hand, learning-based methods utilize machine learning techniques to automate these model generation and parameter tuning steps, either in a supervised setting using expert demonstrations or in a reinforcement learning setting using trial and error. Supervised learning-based approaches are purely data-driven and easy to use, but only work well in static or sparse dynamic environments and require laboriously collecting a representative set of expert demonstrations to train networks [6, 68, 73–76]. Reinforcement learning-based approaches are experience-driven, similar to human learning, and work in crowded dynamic environments, but typically rely solely on simulated data and require carefully designing a reward function [69–72, 77–81, 83–85, 110–114]. Although each type of approach has its own advantages and disadvantages, we choose the reinforcement learning-based framework to address crowded dynamic navigation because it is difficult to manually design a general model or collect effective training data in uncontrolled and human-filled environments. The reward function design also gives us more freedom to integrate the strengths of model-based and learning-based approaches.

However, even with DRL-based frameworks, there are still many open issues that need to be addressed in crowded dynamic navigation. One key set of issues in environmental modeling, representation, and prediction is how to process the sensor data perceived from crowd dynamic environments, how to analyze environmental factors (*e.g.,* static obstacles and moving pedestrians) to predict the future states of the surrounding environments, and how to efficiently and accurately model complex environmental states and design efficient data representations to represent them. Another key set of issues in control policies is how to improve the generalizability and bridge the sim-to-real gap, how

to maintain a balance between different navigation needs (*e.g.,* crowd-aware, intent-aware, uncertainty-aware, and socially-aware), and how to design efficient reward functions for different robot navigation requirements.

# CHAPTER 4
# ENVIRONMENTAL MODELING AND REPRESENTATION

Model-based approaches usually preprocess the raw sensor data and use the extracted high-level information, such as the obstacle costmaps [5, 103], the relative velocity of obstacles [59–61, 104], and the environment dynamics [105, 106]. On the other hand, end-to-end learning approaches focusing on static environments directly feed the raw sensor data, *e.g.,* lidars [68] or cameras [74–76], and goal information into CNNs. DRL-based approaches are more varied, with some also feeding raw lidar scans (and goal points) into CNNs [69–72, 85] while others extract and utilize the position and velocity of pedestrians [78–80], model robot-human interactions [77, 83, 110, 111], predict pedestrian trajectories [81, 84], predict latent states [112], or construct obstacle cost matrices [114]. The long-term success of model-based approaches and the results of the recent DRL studies demonstrate the utility of preprocessed data representations over raw sensor data, as they contain more useful information about pedestrian motion which, in turn, improves safety in dynamic environments. However, these different preprocessed data representations are not specialized for crowded dynamic environments.

Based on these trends, we propose a new set of preprocessed intermediate data representations that are specialized for navigating crowded dynamic spaces and allow good generalization to new environments[1]. Specifically, we first use the camera and 2D lidar as our sensors to perceive the environment. Second, using camera sensor data, we use the YOLOv3 [115] detector and multiple hypothesis tracker (MHT) [116] to detect and

[1]We have open-sourced the data preprocessing code ( `https://github.com/TempleRAIL/drl_vo_nav`) and the 3D human-robot interaction Gazebo simulator code (`https://github.com/TempleRAIL/pedsim_ros_with_gazebo`).

track moving pedestrians and obtain pedestrian kinematics. Third, using lidar sensor data, we explore the use of different lidar preprocessing operations to obtain useful information on static obstacles. Finally, we design efficient data representations to encode useful information about moving pedestrians and static obstacles, which are evaluated in crowded dynamic navigation experiments. Creating abstract data representations that allow us to simplify the world, which reduces the sim-to-real gap since we do not need an exact replication, unlike approaches that use raw data as the input.

## 4.1 Problem Formulation

As shown in Fig. 3.3, a key component of the learning-based control policy is the input data, which encodes and represents observed information about the state of the environment $\mathbf{s}$. In order to autonomously navigate to a goal through a dynamic environment with moving pedestrians, the robot must extract useful information from sensors and process this information to model and represent the state of the environment. The robot then uses this environmental data representation $\mathbf{s}$ to compute the suitable steering velocities via a learning-based control policy, which takes the form of a parametric model:

$$\mathbf{u} = \pi_\theta(\mathbf{s}), \tag{4.1}$$

where $\mathbf{u}$ are the steering velocities and $\theta$ are our policy network parameters. The steering velocities consist of $\mathbf{u} = [v_x, w_z]$, where $v_x$ is the translational velocity and $w_z$ is the rotational velocity in the robot's local coordinate frame.

The environment modeling and representation problem can be formulated as finding the most efficient data representation of the environment state $\mathbf{s}^*$ for optimal control velocities $\mathbf{u}^*$ (*i.e.,* optimal control policy $\pi_\theta^*$):

$$\mathbf{u}^* = \arg\min_{\mathbf{s},\theta} \mathbf{J}(\mathbf{s},\mathbf{u}), \tag{4.2}$$

where $\mathbf{J}(\cdot)$ is the cost function that learning-based approaches aim to minimize.

## 4.2 Preprocessed Data Representation

### *4.2.1 System Overview*

Since the robot's surrounding environment contains static obstacles (*e.g.,* tables, chairs, and walls) and moving pedestrians, we use lidar preprocessed data $\mathbf{l}$ to model and represent the static obstacles, pedestrian kinematics $\mathbf{p}$ to model and represent dynamic pedestrians, and a sub-goal position $\mathbf{g}$ instead of the final goal position to represent the destination. The data representation of environment state can be rewritten as $\mathbf{s} = [\mathbf{l}, \mathbf{p}, \mathbf{g}]$. Note, all data in the state $\mathbf{s}$ is expressed in the local robot frame. This allows our method to be robust to errors in robot localization with respect to a global reference frame, which happens more frequently in densely packed dynamic environments as there are fewer stationary landmarks for the robot to localize itself against. Additionally, relative data is more natural for planning purposes since navigating in a dense crowd is more about going with the flow of traffic than meeting some absolute velocity constraints.

Furthermore, we design a supervised learning-based control strategy to help design and test the effectiveness of the proposed data representation. The control policy network

$\pi_\theta$ is outlined in Fig. 4.1. We combine the three input data feature maps using concatenation fusion, where they are simply stacked along the depth dimension. The concatenated maps are then passed through one 2-D convolutional layer, six bottleneck residual blocks [16] with two block skip connections, and two 2-D pooling layers. Every convolutional layer in our deep neural network is followed by one batch normalization layer and one ReLU action layer. At the end of this early fusion network block, we can extract the high-level features of lidar and pedestrian information. Then, we fuse those high-level features with the goal information in single a fully connected layer. The final steering velocities $\mathbf{u} = [v_x, w_z]$ are generated directly from this fully connected layer without the ReLU action layer. Note that one key feature of our policy network is the use of early fusion to combine three separate feature maps, one representing the lidar data $\mathbf{l}$ and two representing pedestrian kinematics $\mathbf{p}$. Compared with late fusion, early fusion has two obvious advantages [117]. First, early fusion can fully exploit the input information since features are fused at the early stage. Second, it has low memory and computation requirements compared to late fusion, making it more suitable for embedded devices and real-time operation.

To find the policy parameters $\theta$ from expert demonstrations, $\bar{\mathbf{u}}$, we use the mean square error (MSE) between the expert velocity and the learned velocity as our cost function or loss function $\mathbf{J}(\cdot)$:

$$\mathbf{J}(\mathbf{s}, \mathbf{u}) = \frac{1}{N} \sum_{i=1}^{N} \left[\pi_\theta(\mathbf{s}_i) - \bar{\mathbf{u}}_i\right]^2, \tag{4.3}$$

where $N$ is the number of training data tuples $\mathbf{s}_i$ and $\bar{\mathbf{u}}_i$ are the expert examples. We use Adam optimizer [65], a stochastic gradient descent method, to find the optimal model pa-

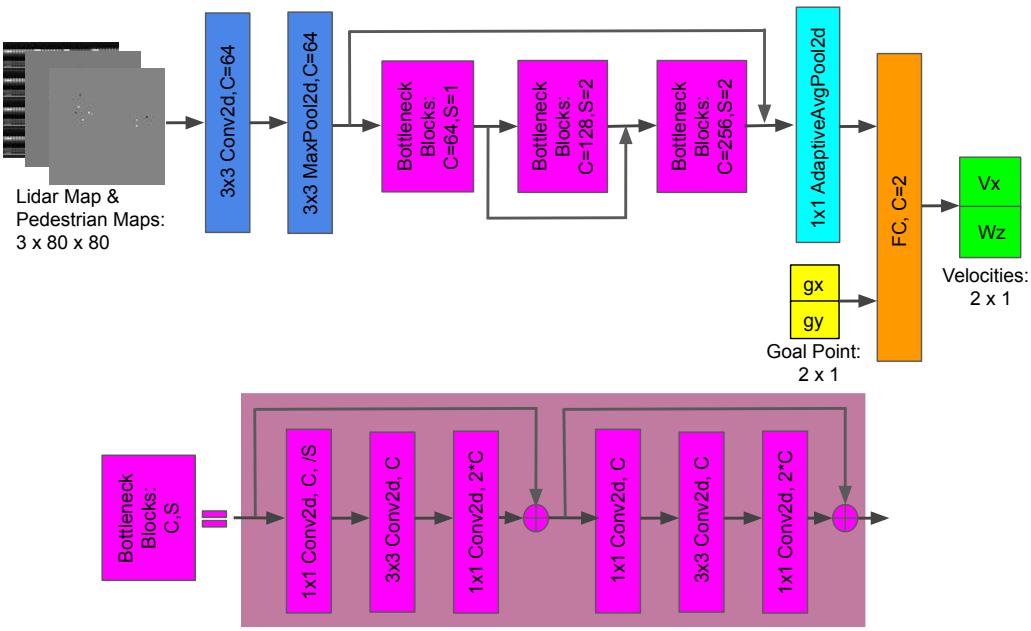
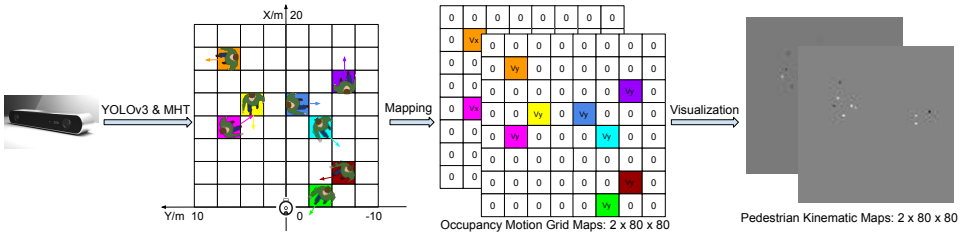



Figure 4.1. Architecture of the proposed CNN deep neural network. Our input data consists of three separate channels, one from lidar and two containing pedestrian data, each of which is an $80 \times 80$ array. These 3-channel input data are fed to the deep neural network consisting of 1 convolutional layer, 6 bottleneck residual blocks [16], and 2 pooling layers. The output of this chain, along with the relative pose of the goal, is fused in a fully connected layer. The resulting outputs of our network are the linear and angular steering velocities. The values "$3 \times 3$" or "$1 \times 1$" in the convolutional layer or pooling layer denote the kernel size. The letters "C" and "S" in the related layers denote the number of output channels and the stride length, respectively.

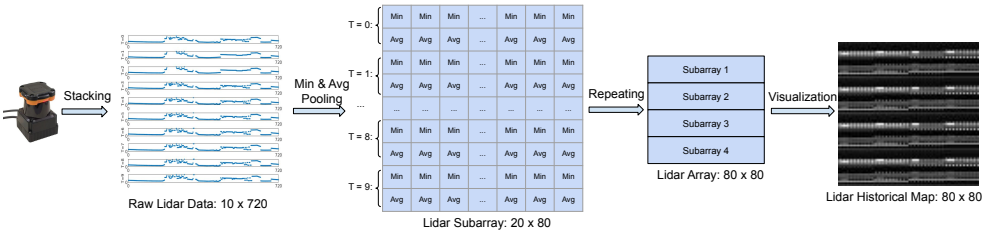
rameters $\theta^*$. We use the step decay with an initial learning rate of $10^{-3}$ as the learning rate schedule and set the mini-batch size to 128 during the training process.

### *4.2.2 Pedestrian Data Preprocessing*

One key contribution of our work is the inclusion of preprocessed data as the input to our CNN control policy. Unlike complex CAGE representations for crowd flow prediction [118], which only encode crowd location information, our proposed simple pedestrian kinematic map representation encodes both the location and velocity information of detected pedestrians. Specifically, to extract information about pedestrians, we first feed the




(a) Pedestrian kinematic maps




(b) Lidar historical map

Figure 4.2. Illustration of the proposed novel pedestrian kinematic maps and lidar historical map. (a) We get estimates of pedestrian kinematics from the YOLOv3 detector and an MHT tracker. We convert the resulting data into two $80 \times 80$ occupancy grid maps, where the pedestrian position determines the cell ID and the velocity gives us the value in that cell. Finally, we standardize the data before feeding them into our network. (b) The input is raw lidar data containing $10 \times 720$ points, since there are 10 scans with 720 measurements per scan. First, we use pooling to downsample each scan to yield a $20 \times 80$ array. Second, we repeat this subarray four times to construct an $80 \times 80$ lidar array. Finally, we normalize the data in this array before feeding it into our CNN network.

raw sensor data into an object detector to extract instantaneous estimates of pedestrian locations. These estimates are fed into a multi-target tracker, which performs data association to yield a collection of target tracks containing position and velocity information. Finally, we put this kinematic data into data structures specifically designed for our network architecture. We hypothesize that this detailed information about the motion of individual pedestrians will enable the robot to more safely navigate. Note, we use a combination of a stereo camera and 2D lidar.

**Object Detector.** To detect pedestrians, we fuse data from the camera and lidar sensors. The camera data is passed into the YOLOv3 detection algorithm [115] to find

labeled bounding boxes for each pedestrian. In parallel, the lidar data is passed into a leg-detection algorithm [119]. We associate the data from these sensors by projecting the laser scan points into the camera image and use this ensemble detector with strong robustness to detect pedestrians and measure their relative positions. Note, we found that the leg detections from the lidar provide a more accurate estimate of the location of each pedestrian while the camera data provides more robust labeling. However, it is possible to use either sensor individually if one desires.

**Object Tracking.** The resulting point estimates of pedestrians are fed into a multiple hypothesis tracker (MHT) [116]. We elect to use the MHT over other multi-target trackers since it was recently shown to achieve state-of-the-art results in the MOT (Multiple Object Tracking) Challenge [120], which involves tracking a large number of people in a crowded scene. One distinction from previous work with the MHT, we perform the tracking in the robot's local coordinate frame instead of a global reference frame. The MHT outputs a list of tracks, each containing the relative position and velocity of a single pedestrian. Note, while the MHT can give us past information about the motion of pedestrians, we only use current information because most of the data is redundant for motion prediction and consequently ignored by neural networks [121]. We use the constant velocity motion model for pedestrians, as Scholler et al. [121] recommend, and update the MHT at a rate of 10 Hz. We also tested the inclusion of both past and future (*i.e.,* predicted) pedestrian data, which either decreased or did not improve the performance.

**Data Structure.** Given that the number of pedestrians in a crowd varies greatly based on both time and location, we cannot simply use the direct output of the MHT as our network architecture requires a uniformly sized input. Instead, we encode the pedestrian

kinematics into occupancy grid-style maps, as Fig. 4.2(a) shows. The basic procedure starts with setting the parameters of the grid. We use an $80 \times 80$ grid with 0.25 m cells, for a total area of $20 \times 20$ m. We selected this cell size as it is small relative to the size of the targets (*i.e.,* people) being tracked, ensuring that each cell in the resulting grid will only have at most one pedestrian in it. We use two simultaneous grids, each one containing information about target velocity in a different direction, here $v_x$ and $v_y$ separately. Note, we use Cartesian instead of polar coordinates as we found these to yield superior performance (results not included due to page limitations).

These $x$ and $y$ velocity grids form our pedestrian kinematic data $\mathbf{p}$. Before this is fed into our early fusion network, we use the standardization procedure:

$$\mathbf{p} = \frac{\mathbf{p} - \mu_{\mathbf{p}}}{\sigma_{\mathbf{p}}}, \tag{4.4}$$

where $\mu_{\mathbf{p}}$ and $\sigma_{\mathbf{p}}$ are the mean and standard deviation of the data in the map. Compared with min-max normalization, standardization does not have a bounding range and is more robust to outliers.

### *4.2.3 Lidar Data Preprocessing*

Most initial works using CNN-based control policies for robot navigation used the entire lidar scan message. However, our early fusion architecture requires that the lidar feature map be the same size as the pedestrian feature maps. Thus, we need to convert the lidar data into an $80 \times 80$ feature map.

One interesting argument proposed by Pfieffer et al. [122] is that using the entire lidar scan causes the resulting network to overfit to the specific environment(s) it was trained in,

as the CNN network learns to extract map features when presented with a dense point cloud. Instead, they use a minimum pooling step to downsample their lidar scan from 1080 data points to 36, with this change leading to better navigation performance in the training environment and better generalizability to other environments. Note that using minimum pooling is a conservative choice, as this will return the nearest lidar return within each section of the scan.

Motivated by this, we explore the use of different lidar preprocessing operations, including reshaping the raw lidar data, projecting scan points into an occupancy grid map, and pooling (minimum, average, median, and maximum). We use each of these different operations, and several combinations thereof, to train our navigation policy in a supervised manner. Figure 4.3(a) shows the resulting mean square error (MSE), *i.e.,* the loss, between the learned and expert velocities for each resulting control policy. We found that using a combination of minimum pooling and average pooling to extract two separate distance measurements per scan region, as Fig. 4.3(a) shows, yielded the best performance. We hypothesize that this combination allows the robot to avoid collisions with nearby obstacles (minimum pooling) while also better understanding the geometry of the free space around it (average pooling). In the end, each individual lidar scan results in 80 minimum pooled values and 80 average pooled values.

Once we determined the most informative way to structure the lidar data, we then studied how long of a history of scans to keep. We considered time windows of 0, 0.25, 0.5, 1, and 2 seconds,[2] using the same training and validation procedure as we did when studying the different lidar preprocessing operations. Figure 4.3(b) shows the resulting MSE loss

[2]Note: we sample the lidar data at 20 Hz, so each time corresponds to a different number of scans.

curve for each resulting control policy. We see that a short history of lidar data contains more useful information than only the current scan (*i.e.,* 0 s), while keeping data from beyond a certain time period resulted in no improvement or even degraded performance. We found the best time window to be 0.5 s (*i.e.,* 10 lidar scans), which we conceptually think of as the effective time constant for pedestrian information.

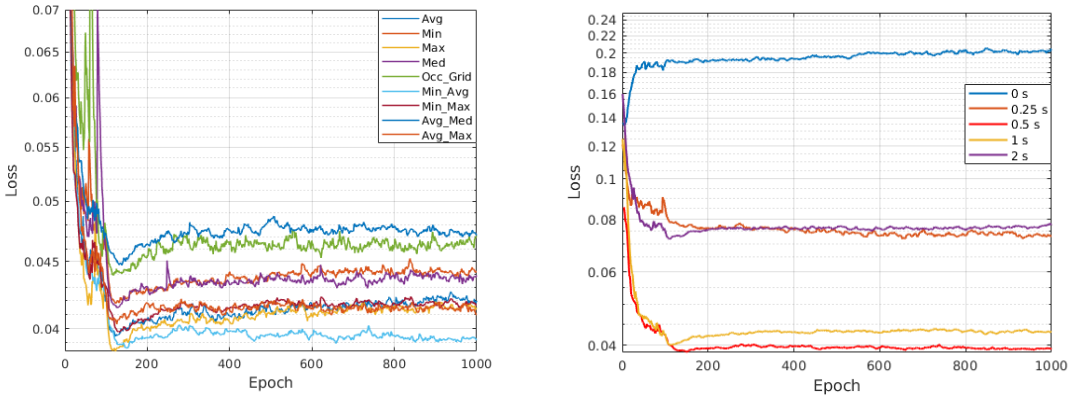
To put the lidar data into a format that is compatible with the pedestrian map so that the two data sources can be fused early, we stack the historical data in a 2D array, using alternating rows of minimum pooling and average pooling. The resulting array is $20 \times 80$, so we then stack four copies of this to create an $80 \times 80$ array of the same size as the pedestrian map. Note, we do not require there to be any spatial correlations between lidar data and pedestrian kinematics data. Also, changing the size of the pedestrian map will also require changes to the parameters of the lidar processing pipeline as the two data sources must be of compatible sizes. Finally, just like the pedestrian data, we apply the standardization procedure prior to feeding the lidar data $\mathbf{r}$ into the early fusion network:

$$\mathbf{l} = \frac{\mathbf{l} - \mu_{\mathbf{l}}}{\sigma_{\mathbf{l}}}, \tag{4.5}$$

where $\mu_{\mathbf{l}}$ and $\sigma_{\mathbf{l}}$ are the average value and standard deviation of the entire scan data.

#### *4.2.4 Goal Position*

Lastly, the robot needs to know the relative position of the goal. Instead of feeding the final goal point to our CNN network, we use the sub-goal point along a nominal path to the final goal as this allows the robot to traverse long paths through complex, non-convex environments and avoid issues with normalization as the distance to the sub-goal is




(a) Lidar preprocessing steps

(b) Lidar history length

Figure 4.3. Validation loss curves of different lidar data configurations. The loss is the MSE between the expert velocity and the learned velocity of the resulting control policy.

constant. To select the sub-goal $\mathbf{g}$, we use the pure pursuit algorithm [123], which calculates the point along the full path that is a specified distance (2 m in our case) ahead of the robot. By tuning this distance, the robot can look far ahead or right nearby. Figure 4.4 illustrates this process. We hypothesize that this sub-goal information will enable the robot to more accurately navigate to the final goal position.

## 4.3 Experiments and Results

Unfortunately, the COVID-19 pandemic has prevented us from conducting real-world experiments because our method is centered around navigation through dense crowds. As a result, we have used Gazebo [124] to simulate the robot and PEDSIM [125], a microscopic pedestrian crowd simulation library based on social forces, to simulate pedestrian motion [3]. This section describes this setup in greater detail, presents the procedure we used to train our network, and compares the results to other methods.

[3]We have open-sourced the 3D Gazebo simulator code at `https://github.com/TempleRAIL/pedsim_ros_with_gazebo`.

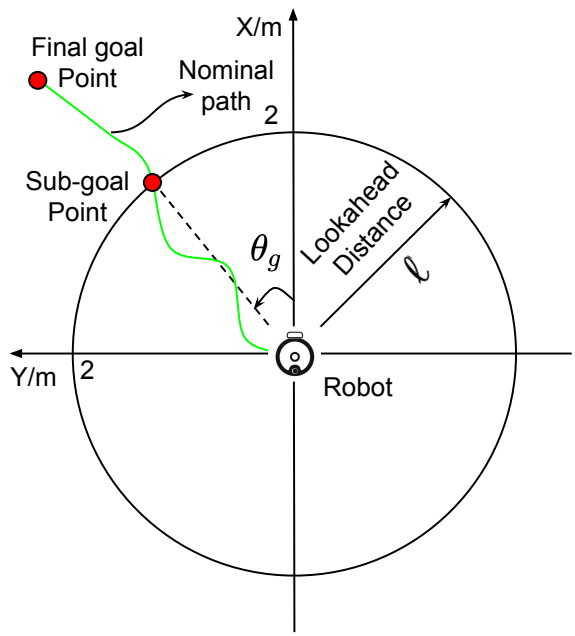


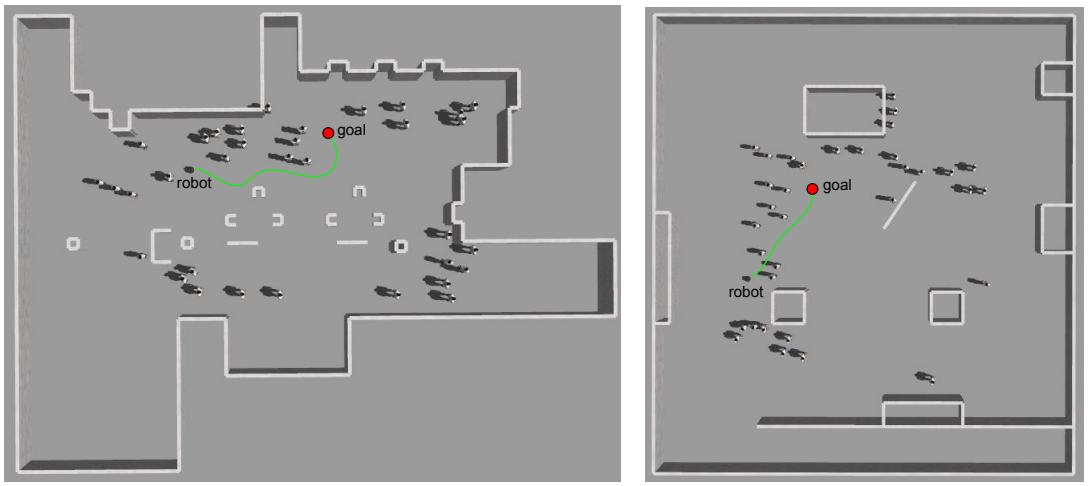

Figure 4.4. Preprocessing step for the sub-goal position. Starting from the nominal path from the robot's current location to its goal (*e.g.,* from A*), we draw a circle of radius $\ell = 2\,\text{m}$ around the robot. The sub-goal point is then where that path first exits the circle in the forward direction.

We use PyTorch to implement our CNN [126]. To train our network, we use the Owl's Nest high-performance computing (HPC) machine learning server, which has 40 CPU cores and 512 GB of RAM, and 8x NVIDIA Tesla Volta V100 GPUs with NVlink2. We run all simulations on a desktop computer with an Intel Core i7-6800K CPU and a GeForce GTX 1080 GPU with 8 GB memory. This computer runs Ubuntu 16.04 and we use ROS Kinetic Kame and Gazebo 9.13.1.

### *4.3.1 Simulation Configuration*

PEDSIM uses the social forces model to guide the motion of individual pedestrians. The basic equation used is

$$\mathbf{F_p} = \mathbf{F_p^{des}} + \mathbf{F_p^{obs}} + \mathbf{F_p^{per}}, \tag{4.6}$$




(a) Lobby world (b) Square world

Figure 4.5. Gazebo simulation environments. The lobby world has two configurations with 34 pedestrians and 50 pedestrians respectively, while the square world only has one configuration with 34 pedestrians. The green curve is the robot trajectory and the red point is the goal point.

where $\mathbf{F_p}$ is the resultant force that determines the motion of a pedestrian; $\mathbf{F_p^{des}}$ pulls a pedestrian towards a destination; $\mathbf{F_p^{obs}}$ pushes a pedestrian away from static obstacles; $\mathbf{F_p^{per}}$ models interactions with other pedestrians (*e.g.,* collision avoidance or grouping). We added another repulsive social force $\mathbf{F_p^{rob}}$ between pedestrians and the robot to model the way people would naturally avoid collisions, thereby allowing our control policy to learn this behavior.

Figure 4.5(a) shows the main environment, a replica of the lobby in the College of Engineering building at Temple University, that we used to train and test our policy. This environment is roughly $25 \times 10$ m in size and is filled with a number of static obstacles (*e.g.,* chairs, tables, a security desk, waste bins, and pillars). The second environment, shown in Fig. 4.5(b), is an artificially created environment that is about $20 \times 20$ m in size. Using these two worlds, we set up three scenarios:

- **Lobby world with 34 pedestrians**: used to collect training data and test basic navigation performance
- **Lobby world with 50 pedestrians**: used to test generalization ability across different crowd densities
- **Square world with 34 pedestrians**: used to test generalization ability to unseen or unknown environments

The robot model is a Kobuki Turtlebot 2, which has a maximum velocity of 0.5 m/s, equipped with a Hokuyo UTM-30LX lidar and a Stereolabs ZED stereo camera. The Hokuyo lidar has a maximum range of 30 m, a FOV of 270°, and an angular resolution of 0.25° while the ZED camera has a minimum range of 0.5 m, a maximum range of 10 m, and a FOV of 90°.

### *4.3.2 Data Collection*

Before we can find the optimal $\theta^*$ that minimizes (4.3), we first need to collect a navigation dataset with expert demonstrations $\bar{\mathbf{u}}$ to train our deep neural network. In the future, once crowds can safely gather again, we plan to collect data from actual humans navigating through crowds. In the simulation, we first tried teleoperating the robot. However, we found this to be very time-consuming and it did not lead to reliable behavior. On the other hand, we found the DWA planner from the ROS navigation stack can achieve relatively good navigation performance in our simulation environment. So as a trade-off result, like many other similar works [68, 122], we use the DWA planner as our expert.

To collect our training data, the robot was repeatedly assigned to reach a random goal within the free space of the map. PEDSIM controlled the behavior of the 34 pedestrians,

which were divided into clusters with a few members each and with each cluster following a series of waypoints to circulate through the environment. During navigation, we recorded the state $\mathbf{s}$ and the expert steering velocities $\bar{\mathbf{u}}$ (*i.e.,* the output of DWA). Every time that the robot collided with an obstacle, we stopped recording data, truncated the data shortly before the collision, and restarted the simulation to avoid learning behavior that would result in unsafe collisions. Although truncating data near a collision may still lead to undesirable behavior, this data is a very small percentage of the total dataset so we do not expect it to have a significant effect on the resulting policy. The final dataset contains 157,809 $(\mathbf{l}, \mathbf{p}, \mathbf{g}, \bar{\mathbf{u}})$ tuples, which we divided into three separate subsets for training (103,121 tuples), validation during training (22,349 tuples), and final testing (32,339 tuples).

### *4.3.3 Training Results*

Using the training dataset from Sec. 4.3.2, we trained three neural networks to demonstrate the efficacy of our novel early fusion neural network with pedestrian motion information. The first neural network is a state-of-the-art, end-to-end network that uses raw lidar data and the relative goal position as its input [68]. The second one is our architecture using only lidar data and the relative goal, to make a direct comparison with Pfeiffer et al. [68]. The last one is our full architecture, including the pedestrian data.

As in Loquercio et al. [75], we use the following two metrics to evaluate the regression performance of each network:

- **Root mean square error (RMSE)**:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{\mathbf{u}}_i - \mathbf{u}_i)^2}, \tag{4.7}$$

Table 4.1. Training results on the robot navigation datasets.

| Method | RMSE | EVA | # of Params | FPS |
|---|---|---|---|---|
| Pfeiffer et al. [68] | 0.4265 | 1.0419 | 50.732 M | **696.675** |
| Lidar | 0.1794 | 0.1774 | **28.925 M** | 266.982 |
| Lidar + Pedestrian | **0.1673** | **0.1543** | 28.939 M | 255.127 |

- **Explained variance ratio (EVA)**:

$$EVA = \frac{\sum_{i=1}^{N} [(\bar{\mathbf{u}}_i - \mathbf{u}_i) - \mu_{\bar{\mathbf{u}}_i - \mathbf{u}_i}]^2}{\sum_{i=1}^{N} (\bar{\mathbf{u}}_i - \mu_{\bar{\mathbf{u}}})^2}. \tag{4.8}$$

Figure 4.6 shows the validation loss curves for each network during training and Table 4.1 shows the final training results after 3000 epochs. We can observe that although our novel network architecture is nearly two times smaller than Pfeiffer's network [68], it yields significantly better regression performance with smaller RMSE and EVA. Furthermore, our network with pedestrian motion information performs better than only using lidar data, demonstrating that pedestrian motion can help train the deep neural network. Finally, due to the use of early fusion, the addition of pedestrian data has a negligible effect on the number of parameters (only a 0.05% increase). All three methods are clearly capable of real-time processing, with average frames per second (FPS) of over 250, far above the 40 Hz of the sensors.

### *4.3.4 Navigation Results*

While achieving superior loss during training is encouraging, we ultimately care about the resulting behavior of our policy during navigation. We test each of the poli-

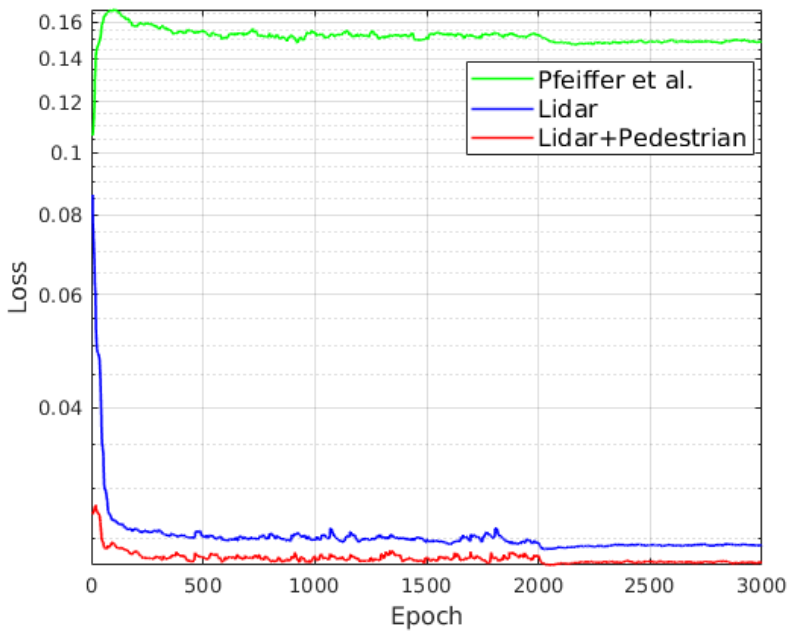
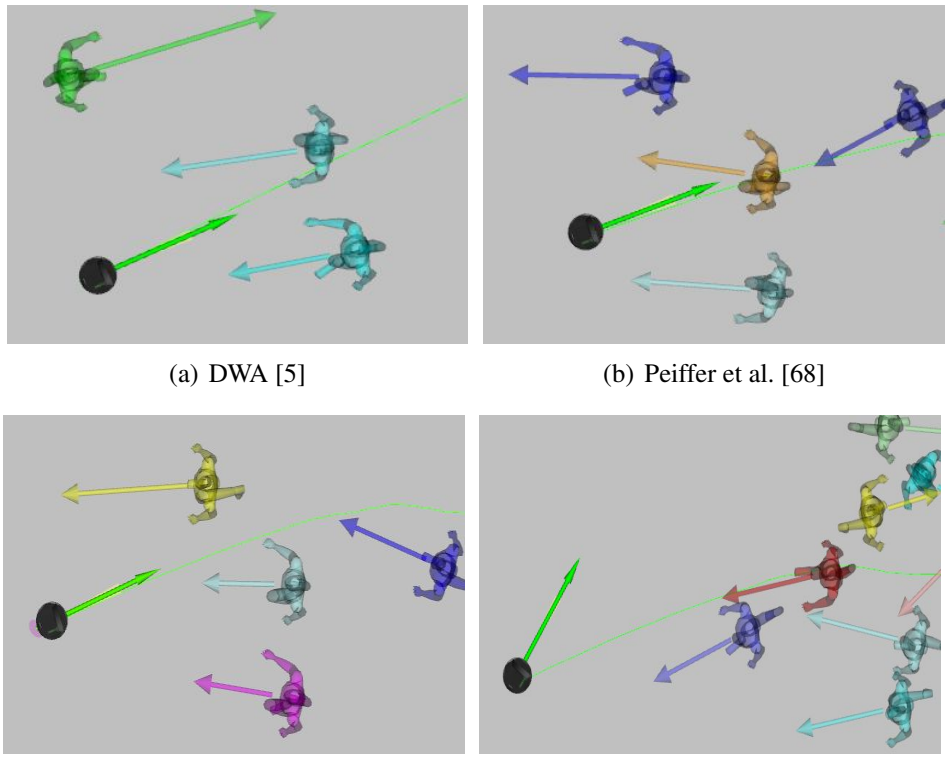



Figure 4.6. Validation loss curves during the training process. Note that all three networks converge to stable values.

Table 4.2. Navigation results in different simulated environments.

| Environment | Method | Success Rate | Average Time (s) | Average Length (m) | Average Speed (m/s) |
|---|---|---|---|---|---|
| Lobby world with 34 pedestrians | DWA [5] | 0.79 | **12.468** | **5.164** | **0.414** |
| | Pfeiffer et al. [68] | 0.73 | 17.549 | 5.589 | 0.318 |
| | Lidar | 0.78 | 14.788 | 5.483 | 0.371 |
| | Lidar + Pedestrian | **0.84** | 15.732 | 5.598 | 0.356 |
| Lobby world with 50 pedestrians | DWA [5] | 0.67 | **13.586** | **5.164** | **0.380** |
| | Pfeiffer et al. [68] | 0.64 | 18.203 | 5.858 | 0.322 |
| | Lidar | 0.68 | 14.635 | 5.411 | 0.370 |
| | Lidar + Pedestrian | **0.78** | 18.229 | 5.823 | 0.319 |
| Square world with 34 pedestrians | DWA [5] | 0.72 | **21.319** | **8.601** | **0.403** |
| | Pfeiffer et al. [68] | 0.54 | 57.979 | 10.826 | 0.187 |
| | Lidar | 0.62 | 29.950 | 9.968 | 0.333 |
| | Lidar + Pedestrian | **0.75** | 26.370 | 9.330 | 0.354 |

cies from Sec. 4.3.3, along with the DWA planner [5], in each of the three scenarios from Sec. 4.3.1. We compare the performance using the following metrics, which are commonly used in the autonomous navigation literature [70, 84, 111]:



(a) DWA [5]

(b) Peiffer et al. [68]

(c) Lidar

(d) Lidar + Pedestrian

Figure 4.7. Robot reactions to moving pedestrians using different control policies. Each figure shows the pedestrians (colored figures), the current velocity of each pedestrian (colored arrows), the robot (black circle), the nominal path to the goal (green line), and the computed velocity (green arrow). When the robot encounters a group of pedestrians, (a) DWA [5], (b) Pfeiffer's policy [68], and (c) our lidar-only CNN are more likely to move forward along the nominal path and ignore those pedestrians until the last moment. (d) On the other hand, our CNN-based policy with pedestrian information reacts earlier.

- **Success rate**: the fraction of collision-free trials
- **Average time**: the average travel time of trials
- **Average length**: the average trajectory length of trials
- **Average speed**: the average speed during trials.

In each scenario, we run 4 tests from the same initial conditions, where each test consists of the robot navigating through the same sequence of 25 goal points. Although the

initial conditions of each test are the same, each trial yields different navigation behavior due to sensor noise, social force interactions, etc. Table 4.2 summarizes our results. See the accompanying video for a demonstration of the safe navigation behavior of our CNN-based control policy.

We first focus on the original environment used for testing, the lobby world with 34 pedestrians, where we observe several interesting phenomena. First, our novel CNN-based policies with preprocessed data have a higher success rate and a higher average speed than Pfeiffer's policy [68], which uses raw lidar data. This demonstrates that the combination of model-based and learning-based approaches can yield better navigation performance than a purely learning-based approach. Second, despite the fact that our proposed CNN policy with lidar-only input has similar training results and average speed to our policy with pedestrian data, the latter has a significantly higher success rate. This shows that pedestrian information is useful and plays a key role in enabling safe navigation in crowded and dynamic environments.

Our proposed CNN-based policy with pedestrian motion data has a higher success rate than the DWA planner [5], albeit with a lower average speed. We believe there are two main reasons for this. First, because we removed all collisions from our training dataset, our proposed network only learned the *safe* navigation behaviors from the expert (DWA). Second, pedestrian kinematics help our network understand the motion of pedestrians and learn safe steering behaviors better. Figure 4.7 demonstrates this, where we can see that our proposed CNN-based policy with pedestrian motion data is more likely to change its heading direction early to avoid pedestrians. As a result of this early avoidance behavior, our CNN-based policy travels a longer distance and takes longer to reach the goal. How-

ever, we believe this is worth doing to prevent collisions and that this slowdown could be reduced with better expert data.

The only observable trend in the failure cases of our controller was a slightly increased rate of collisions near the goal location or in particularly dense crowds, though neither occurred regularly. We hypothesize that collisions in dense crowds were more common due to two factors: 1) we used the DWA planner, which fails more often in dense crowds, as our expert, and 2) the social force model breaks down in these situations, with pedestrians often bumping into one another. Future work will aim to mitigate this issue by using an experienced human expert to train our system and real pedestrians to train/test our system (once the COVID-19 pandemic subsides and it is again safe to conduct real-world experiments with dense crowds of people).

Table 4.2 also shows that our policy generalizes better to new environments and crowd sizes than the other approaches, particularly the lidar-only CNNs. Notably, the difference in the success rate between our CNN policy and that of Pfeiffer et al. [68] is much larger in the more crowded environment and the unseen environment than that in the training environment. These results demonstrate that our CNN-based policy is able to more easily transfer the learned navigation policies to the more crowded (with almost 50% more people) environments and unseen environments. Despite these improvements, the generalizability of our policy is still limited, with the robot missing the goal or moving erratically in some unfamiliar scenarios. Future work will mitigate this issue by training our CNN-based policy in different environments and with different crowd sizes.

## 4.4 Conclusion

In this chapter, we proposed a new combination of preprocessed data representations, which can work well in crowded dynamic environments with up to 55 pedestrians and bridge the appearance gap between an imperfect simulation and reality. Specifically, we use a short history of lidar data and the current kinematics of pedestrians to model and represent the environment state. The efficacy of our proposed preprocessed data representations is demonstrated in different simulation experiments with a novel CNN-based control policy trained in a supervised learning manner. The extensive training and navigation experiments demonstrate that the data representation of two pedestrian kinematic maps and one lidar historic map plays a key role in dynamic navigation and that our CNN-based policy has a significantly higher success rate than standard model-based planners or CNN policies with only sensor data. Furthermore, this new set of environment representations makes our proposed CNN-based policy generalize better to different crowd sizes and unseen dynamic environments. Finally, this new set of environment representations is in the robot's local frame, making it robust to errors in localization that are common in complex, crowded, and dynamic environments.

# CHAPTER 5
# CROWD-AWARE CONTROL POLICY

One of the biggest challenges in designing control policies for crowded dynamic scenes using DRL is designing an appropriate reward function. Most approaches include two terms, one to reward progress towards the goal and one to penalize collisions [69–72, 77–81, 83–85, 110–112]. Although this simple reward function design can work well, their sparse collision avoidance reward function only gives a large negative reward to penalize collisions, and does not give a direct and effective reward signal to guide the robot to actively avoid obstacles. In fact, the passive collision avoidance reward function is a zero-order function, meaning it assumes the risk of collision depends only on the distance to the obstacle. We argue that the *relative motion* of pedestrians (and other objects) is more important since people often follow one another closely in dense crowds and give more space when walking in opposite directions. One attempt at this is the reverse reward function from [114], which prioritizes safety but leads to slow motion and can cause deadlock in dense crowds. Clearly, using their reward functions, these DRL-based control policies struggle to maintain a good balance between collision avoidance and speed, especially in crowded dynamic environments.

To address this problem, we propose a novel crowd-aware control policy (*i.e.,* DRL-VO) by designing a novel reward function [1]. Specifically, we first use the handcrafted intermediate data representations from Chapter 4 as the input data. Second, we propose a novel reward function that uses first-order velocity obstacles to penalize headings that are likely

[1]We have open-sourced the crowd-aware navigation code at `https://github.com/TempleRAIL/drl_vo_nav`. Multimedia is also available at `https://www.youtube.com/watch?v=KneELRT8GzU&list=PLouWbAcP4zIvPgaARrV223lf2eiSR-eSS`.

to lead to collisions, thereby guiding the robot to continuously tune its heading direction to actively avoid crowds of pedestrians while making progress towards the goal. Finally, we conduct a set of simulated and real-world navigation experiments to demonstrate the performance of our DRL-VO control policy in crowded dynamic navigation.

## 5.1 Problem Formulation

Due to the limited field of view (FOV) of sensors (*e.g.,* lidars and cameras), there is no complete environmental perception for the robot in the real world. Extracting and processing useful information from sensors to obtain the partial observation of the environment $\mathbf{o^t}$ is the first step for the robot to autonomously and quickly navigate through complex and human-filled environments. The robot then feeds this partial observation $\mathbf{o^t}$ into a control policy $\pi_\theta$ to compute the suitable steering action $\mathbf{a^t}$, which is defined as

$$\mathbf{a^t} \sim \pi_\theta(\mathbf{a^t}|\mathbf{o^t}), \tag{5.1}$$

where $\theta$ are our control policy parameters. This complicated navigation decision-making process can be formulated as a partially observable Markov decision process (POMDP), which is denoted by a 6-tuple $\langle S, A, T, R, \Omega, O \rangle$ where $S$ is the state space, $A$ is the action space, $T$ is the state-transition model, $R$ is the reward function, $\Omega$ is the observation space, and $O$ is the observation probability distribution. One of the most well-known tools for solving large POMDPs is deep reinforcement learning [127]. The general objective $L(\theta)$

of DRL is to maximize the expected discounted return by optimizing the neural network-based policy $\pi_\theta$:

$$L(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R^t\right], \tag{5.2}$$

where $\gamma$ is the discount rate in the range of $[0,1]$, $\mathbb{E}_{\pi_\theta}[\cdot]$ denotes the expected value of a random variable assuming the agent follows the neural network-based policy $\pi_\theta$.

## 5.2 DRL-based Control Policy

Figure 5.1 outlines our DRL network, which consists of four modules: feature generation, feature extractor, actor, and critic. The feature generation module generates our intermediate features, *i.e.,* the partial observation $\mathbf{o}^t$, from the raw sensor data. The feature extractor module then extracts high-level features from $\mathbf{o}^t$, the critic module generates the state value, and the actor module generates steering action $\mathbf{a}^t$.

### *5.2.1 Observation Space*

We use the same partial observation $\mathbf{o}^t = [\mathbf{l}^t, \mathbf{p}^t, \mathbf{g}^t]$ as Chapter 4, which consists of three preprocessed components: lidar history ($\mathbf{l}^t$), pedestrian kinematics ($\mathbf{p}^t$), and the sub-goal position ($\mathbf{g}^t$). All of the data in the observation $\mathbf{o}^t$ are expressed in the local reference frame of the robot. This allows our method to be robust to errors in robot localization with respect to a global reference frame, which happens more frequently in densely-packed dynamic environments as there are fewer stationary landmarks for the robot to localize itself against. Additionally, relative data is more natural for planning purposes since navigating in a dense crowd is more about going with the flow of traffic than meeting some absolute velocity constraints. Note, one key difference between our work and most other learning-

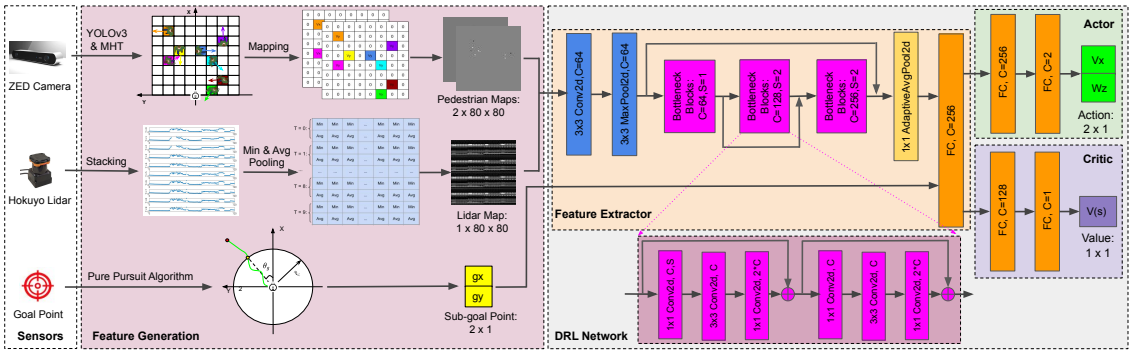



Figure 5.1. Overall system architecture of the proposed DRL-VO control policy. Raw sensor data from the ZED camera and Hokuyo lidar, as well as the goal point, are fed into a feature generation module to create intermediate features. Three $80 \times 80$ intermediate-level feature maps, one from lidar and two containing pedestrian information, along with the relative position of the sub-goal are fed to the feature extractor network, and the resulting high-level features are fed to the actor and critic networks separately. The critic network outputs the state value and the actor network outputs the linear and angular steering velocities. The feature extractor module consists of several bottleneck residual blocks [16], while the critic and actor modules consist of fully connected layers. "$3 \times 3$" or "$1 \times 1$" denote the kernel size, and "C" and "S" denote the number of output channels and the stride length, respectively.

based strategies is the use of preprocessed data (as opposed to raw sensor data) as the input to our DRL control policy.

Before these observation data are fed into our control policy network, we use the Min-Max scaling procedure to normalize them to $[-1, 1]$:

$$\mathbf{o}^t = 2\frac{\mathbf{o}^t - \mathbf{o}^t_{\text{min}}}{\mathbf{o}^t_{\text{max}} - \mathbf{o}^t_{\text{min}}} - 1, \tag{5.3}$$

where $\mathbf{o}^t_{\text{max}}$ and $\mathbf{o}^t_{\text{min}}$ are the maximum value and minimum value of the data (*e.g.,* for the lidar history $\mathbf{l}^t$, they are the minimum and maximum range value of the lidar sensor).

### 5.2.2 Action Space

The action $\mathbf{a}^t = [v_x^t, \omega_z^t]$ of our DRL control policy has two terms, the forward ($v_x^t$) and rotational ($\omega_z^t$) velocities in the local robot frame. Note, we use a continuous action space as we believe this gives the robot more accurate control. We limit the range of the forward velocity $v_x^t$ to $[0, 0.5]$ m/s and the rotational velocity $\omega_z^t$ to $[-2, 2]$ rad/s based on the limitations of the Turtlebot2, our target hardware platform. Note that similar to observation space, we also normalize the action space to [-1, 1] using the Min-Max scaling procedure.

### 5.2.3 Network Architecture

As shown in Fig. 5.1, the feature generation module contains the YOLO and MHT sequence for generating the pedestrian maps (Sec. 4.2.2), the pooling and stacking operations for generating the lidar map (Sec. 4.2.3), as well as the pure pursuit algorithm for generating the sub-goal (Sec. 4.2.4). The feature extractor module is used to extract high-level features from the intermediate-level partial observation $\mathbf{o}^t$, the critic module generates the state value, and the actor module generates steering action $\mathbf{a}^t$. The feature extractor network is identical to the backbone CNN network of our previous work [6], which fuses the lidar historical observation $\mathbf{l}^t$ and pedestrian kinematics observation $\mathbf{p}^t$. After extracting high-level features, a single fully connected layer is used to fuse those high-level features with the sub-goal information. Both the actor module and the critic module are simply constructed from two fully connected layers. Note that each convolutional layer in our feature extractor module is followed by a batch normalization layer and a ReLU action layer. Additionally, the last fully connected layer in our feature extractor is followed by a ReLU

Table 5.1. Ablation experiment results with different fusion structures.

| **Network Structures** | **RMSE** | **EVA** | **# of Params** | **FPS** |
|---|---|---|---|---|
| Early fusion | **0.17** | **0.15** | **28.94 M** | **255.13** |
| Late fusion | 0.18 | 0.17 | 57.86 M | 140.53 |

action layer while the last fully connected layers of the actor and critic modules have no ReLU action layers.

Our architecture is an example of a middle fusion network, as data from different input streams are fused in the middle of the chain between raw data input and output. Feng et al. [117] provide a detailed discussion about different fusion architectures (in a general setting), which are typically categorized as early, middle, or late fusion. They note that there is no solid evidence to support which fusion network is the best choice, though there is a connection between the computational load of a neural network and its structure as the number of layers and their sizes affect both the number of computations that must be done to produce and output and the memory required to store the parameters of the neural network. To examine this, we used the same dataset and supervised training manner as our previous work [6] to train two networks. The first, which we call Early, fuses the pedestrian kinematics and lidar data at the input to the feature extractor block, as we do above, and the second, called Late, fuses all data just before the output of the feature extractor block. As we can see in Table 5.1, the Early network is about half the size of the Late network, which nearly doubles the processing rate and yields slightly better regression performance, with smaller root mean square error (RMSE) and explained variance ratio (EVA).

Like our network, most other neural network-based control policies [72, 73, 84, 85] also use late or middle fusion architectures as this allows for flexible design because the different input sources each yield some learned intermediate representations which are all combined downstream. The key difference between these and our architecture is that we use handcrafted features as our intermediate representation, much like a traditional model-based algorithm, rather than learned features, as are typical in learning-based algorithms. One benefit of this approach is that our data representations, *i.e.,* the pedestrian map, are independent of the specific object detection and tracking tools used. This allows us to easily swap out YOLOv3 and the MHT for different approaches with no need for retraining.

### *5.2.4 Reward Function*

An essential problem of reinforcement learning methods is how to design a good reward function to guide the agent to learn desired behaviors. Navigation tasks have two competing objectives, we want the robot to move as quickly as possible to reach the goal in minimum time but also safely to avoid colliding with any stationary or moving objects. Thus, we design a multi-objective reward function:

$$r^t = r_g^t + r_c^t + r_w^t + r_d^t, \tag{5.4}$$

where $r_g^t$ rewards making progress towards the goal, $r_c^t$ penalizes passively approaching or colliding with an obstacle, $r_w^t$ penalizes rapid changes in direction, and $r_d^t$ rewards actively steering to avoid obstacles and point towards the sub-goal.

Note that most previous works [69–72, 77–81, 83–85, 110–112] only use such a zero-order passive collision avoidance reward $r_c^t$, a position-based heuristics, to guide robots to passively avoid obstacles.

While these standard terms are useful for learning good navigation behavior, we want to allow robots to indirectly learn from successful model-based navigation policies. To do this, we introduce a new reward term $r_d^t$ that uses velocity obstacles [59] to set a desired navigation direction. This term allows the robot to use first-order kinematic information (*i.e.,* velocity) to proactively avoid collisions while making progress towards the goal.

**Reaching the Goal.** The reward is given by

$$r_g^t = \begin{cases} r_{\text{goal}} & \text{if } \|p_g^t\|) < g_m \\ -r_{\text{goal}} & \text{else if } t \geq t_{\max} \\ r_{\text{path}}(\|p_g^{t-1}\| - \|p_g^t\|) & \text{otherwise,} \end{cases} \tag{5.5}$$

where $p_g^t$ is the goal position (in the robot's frame) at time $t$. We use $r_{\text{goal}} = 20$, $r_{\text{path}} = 3.2$, $g_m = 0.3\,\text{m}$, and $t_{\max} = 25\,\text{s}$.

**Passive Collision Avoidance.** The reward is given by

$$r_c^t = \begin{cases} r_{\text{collision}} & \text{if } \|p_o^t\| \leq d_r \\ r_{\text{obstacle}}(d_m - \|p_o^t\|) & \text{else if } \|p_o^t\| \leq d_m \\ 0 & \text{otherwise,} \end{cases} \tag{5.6}$$

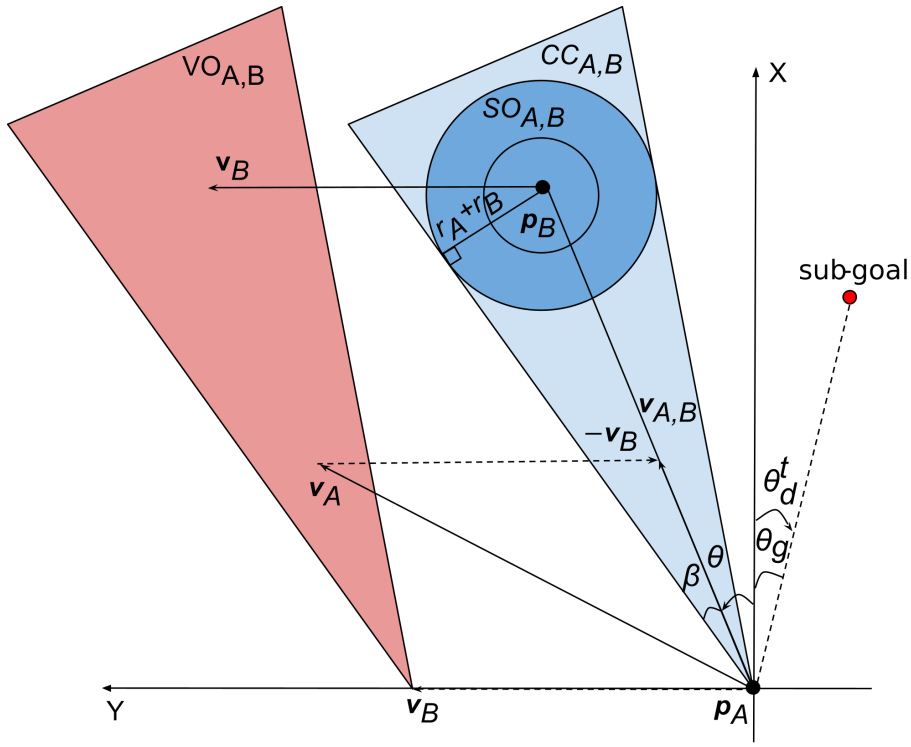



Figure 5.2. The pictogram for velocity obstacle $VO_{A,B}$ (red cone), collision cone $CC_{A,B}$ (light blue cone), and special occupancy $SO_{A,B}$ (blue circle). $A$ represents the robot with radius $r_A$, position $\mathbf{p}_A$, and velocities $\mathbf{v}_A$. $B$ represents the moving pedestrian with radius $r_B$, position $\mathbf{p}_B$, and velocities $\mathbf{v}_B$. $\mathbf{v}_{A,B} = \mathbf{v}_A - \mathbf{v}_B$ represents the relative velocity of $A$ with respect to $B$. Note that all quantities are in the robot's local frame.

where $p_o^t$ is obstacle position at time $t$. We use $r_{\text{collision}} = -20$, $r_{\text{obstacle}} = -0.2$, $d_r = 0.3\,\text{m}$, and $d_m = 1.2\,\text{m}$.

**Path Smoothness.** The reward is given by

$$r_w^t = \begin{cases} r_{\text{rotation}}|\omega_z^t| & \text{if } |\omega_z^t| > \omega_m \\ 0 & \text{otherwise,} \end{cases} \tag{5.7}$$

where $r_{\text{rotation}} = -0.1$ and $\omega_m = 1\,\text{rad/s}$.

**Active Heading Direction.** The reward is given by

$$r_d^t = r_{\text{angle}}(\theta_m - |\theta_d^t|), \tag{5.8}$$

where $\theta_d^t$ is the desired heading direction in the robot's local frame and $\theta_m$ is the maximum allowable deviation of the heading direction. We use $r_{\text{angle}} = 0.6$ and $\theta_m = \frac{\pi}{6}$ rad.

The key to this reward term is to find the desired direction $\theta_d^t$ that moves towards the goal while also being collision-free. To do this, we extend the concept of velocity obstacles [59]. The velocity obstacle $VO_{A,B}$ is the velocity space where the current velocity of robot $A$ would cause a collision with an obstacle $B$ at some future time, as Fig. 5.2 shows. To define the velocity obstacle $VO_{A,B}$, we first need to define the special occupancy $SO_{A,B}$ and collision cone $CC_{A,B}$. The special occupancy $SO_{A,B}$ is a open disc area with radius $r_A + r_B$ centered at $\mathbf{p}_B$, which is defined as

$$SO_{A,B} = \{\mathbf{p}_S \mid d(\mathbf{p}_S, \mathbf{p}_B) < r_A + r_B\}, \tag{5.9}$$

where $d(\mathbf{p}_S, \mathbf{p}_B)$ is the distance between $\mathbf{p}_S$ and $\mathbf{p}_B$. Then, the collision cone is defined as:

$$CC_{A,B} = \left\{\mathbf{v}_{A,B} \mid \exists t, \mathbf{v}_{A,B}t \cap SO_{A,B} \neq \emptyset\right\}. \tag{5.10}$$

Finally, the velocity obstacle $VO_{A,B}$ is defined as:

$$VO_{A,B} = CC_{A,B} \oplus \mathbf{v}_B. \tag{5.11}$$

where $\oplus$ denotes the Minkowski vector sum operator.

The physical meaning of $CC_{A,B}$ is that any relative velocities $\mathbf{v}_{A,B} \in CC_{A,B}$ will cause a collision at a future time. From the perspective of the direction angle of $\mathbf{v}_{A,B}$, the collision cone $CC_{A,B}$ can also be defined as

$$CC_{A,B} \in [\theta - \beta, \theta + \beta], \tag{5.12}$$

where $\theta$ and $\beta$ can be easily calculated from Fig. 5.2 using simple geometric relationships:

$$\theta = \arctan 2 \left( \frac{\mathbf{p}_{B_y}}{\mathbf{p}_{B_x}} \right), \tag{5.13}$$

$$\sin \beta = \frac{r_A + r_B}{\|\mathbf{p}_B\|}. \tag{5.14}$$

Using the collision cone $CC_{A,B}$, the robot can know which heading direction angles will cause collisions with all of the moving pedestrians tracked by MHT. The robot then uses these collision cones and the direction of the sub-goal ($\theta_g$) to find the desired heading direction angle $\theta_d^t$ using Algorithm 5.1. Note that this sampling-based search algorithm is motivated by [60].

### *5.2.5 Deep Reinforcement Learning Algorithm*

While there are many deep reinforcement learning algorithms, we use the proximal policy optimization (PPO) algorithm [28] to train our DRL network. A major advantage of the PPO algorithm is that it is much simpler to implement and can achieve comparable or better results than other state-of-the-art algorithms (*e.g.,* A2C [26], TRPO [27] and

**Algorithm 5.1:** Search desired direction angle

**Input:** Sub-goal direction angle $\theta_g$

**Input:** pedestrians from MHT $B_{\text{peds}}$

**Input:** robot linear velocity $\mathbf{v}_{A_x}$

**Input:** number of samples $N$

**Output:** Optimal direction angle $\theta_d^t$

```
Initialize: θ_d^t ← π/2
if B_peds ≠ ∅ then
    θ_min ← ∞
    for i = 1, 2, ..., N do
        θ_u ← sample from [−π, π]
        free ← True
        for B in B_peds do
            θ_{v_{A,B}} ← atan2( (v_{A_x} sin(θ_u) − v_{B_y}) / (v_{A_x} cos(θ_u) − v_{B_x}) )
            θ, β ← from (5.13), (5.14) using B
            if θ_{v_{A,B}} ∈ [θ − β, θ + β] then
                free ← False
                break
            end if
        end for
        if free then
            if ‖θ_u − θ_g‖ < θ_min then
                θ_min ← ‖θ_u − θ_g‖
                θ_d^t ← θ_u
            end if
        end if
    end for
else
    θ_d^t ← θ_g
end if
return θ_d^t
```

ACER [29]). Furthermore, many other autonomous robot navigation works [69–71, 114] use PPO and have found that it works well on navigation problems. As suggested by [28], we use Adam optimizer [65], a stochastic gradient descent method, to find the optimal policy parameters $\theta^*$.

## 5.3 Crowded Dynamic Environments

To demonstrate the efficacy and performance of our proposed control policy, we first use a Gazebo simulator [124] and the PEDSIM library [125] to conduct a set of 3D simulated experiments, and then use a Turtlebot2 robot to conduct the real-world experiments. This section describes the experimental setup, training procedure, and results.

### *5.3.1 Experimental Setup*

**Robot Configuration.** For the simulated and hardware tests, we use a Turtlebot2 robot with a ZED stereo camera and a Hokuyo UTM-30LX lidar, as shown in Fig. 3.1. The maximum velocity of the Turtlebot2 is limited to 0.5 m/s. The depth range of the ZED camera is set to $[0.3, 20]$ m, and its FOV is $90^\circ$. The measurement range of Hokuyo lidar is set to $[0.1, 30]$ m, its FOV is $270^\circ$, and its angular resolution is $0.25^\circ$.

**Pedestrian Configuration.** PEDSIM is a microscopic pedestrian crowd simulation library, which uses the social forces model [107, 128] to guide the motion of pedestrians:

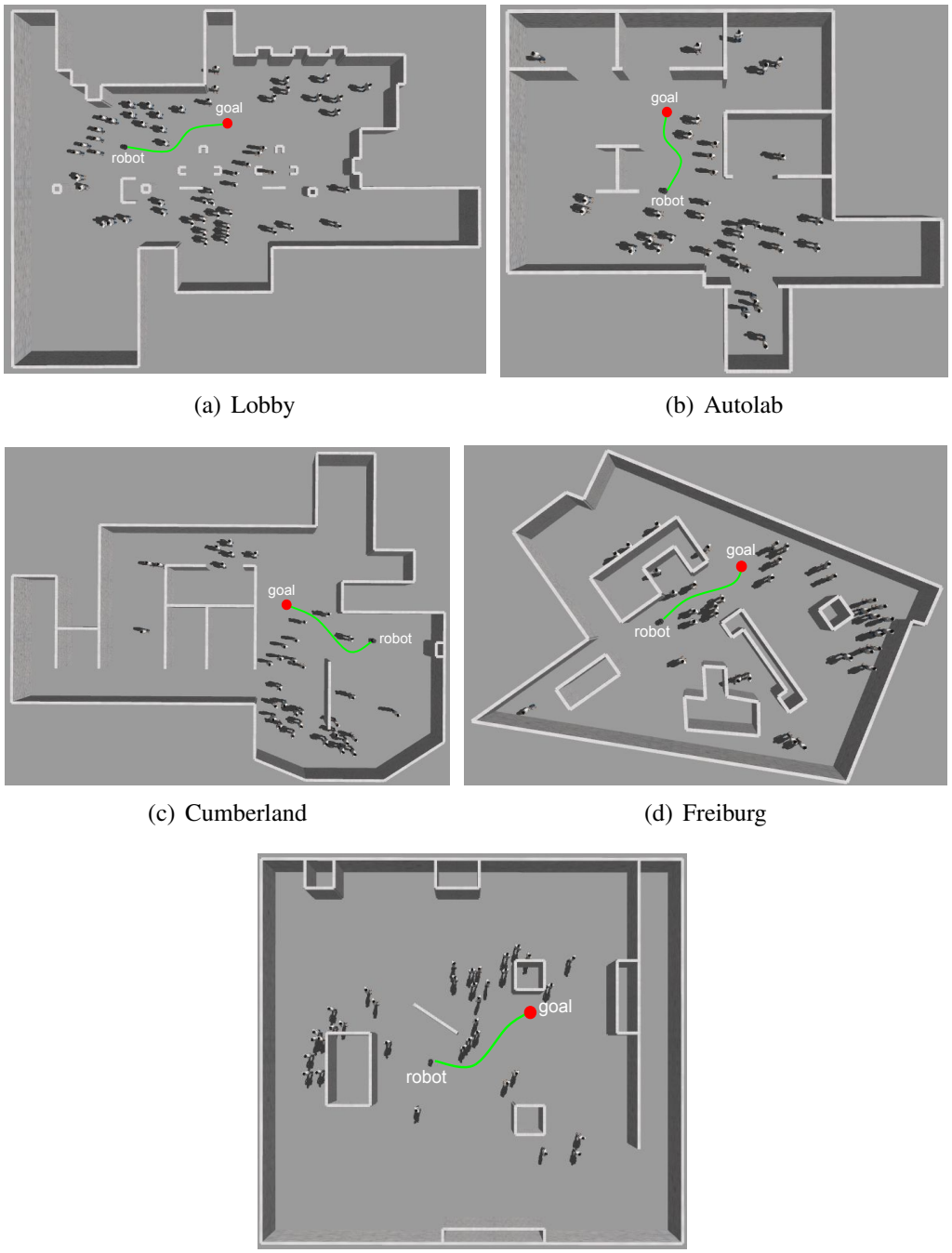
$$\mathbf{F}_p = \mathbf{F}_p^{\text{des}} + \mathbf{F}_p^{\text{obs}} + \mathbf{F}_p^{\text{per}} + \mathbf{F}_p^{\text{rob}}, \tag{5.15}$$

where $\mathbf{F}_p$ is the resultant force that determines the motion of a pedestrian $p$; $\mathbf{F}_p^{\text{des}}$ pulls a pedestrian towards a destination; $\mathbf{F}_p^{\text{obs}}$ pushes a pedestrian away from static obstacles; $\mathbf{F}_p^{\text{per}}$

models interactions with other pedestrians (*e.g.,* collision avoidance or grouping); and $\mathbf{F}_p^{\text{rob}}$ pushes pedestrians away from the robot, modeling the way people would naturally avoid collisions and thereby allowing our control policy to learn this behavior. The first three terms are part of the standard model while the last term is new.

**Simulation Configuration.** Figure 5.3(a) shows the main Gazebo simulation environment, a replica of the lobby in the College of Engineering building at Temple University, that we used to train and test our control policies. This environment is roughly $25 \times 10$ m in size and is filled with a number of static obstacles (*e.g.,* chairs, tables, a security desk, waste bins, and pillars). The second environment, shown in Fig. 5.3(b), is a replica of the Autolab building that is about $20 \times 10$ m in size. The third environment, shown in Fig. 5.3(c), is a replica of the Cumberland building that is about $20 \times 10$ m in size. The fourth environment, shown in Fig. 5.3(c), is a replica of the Freiburg building that is about $20 \times 10$ m in size. The fifth environment, shown in Fig. 5.3(e), is an artificially created environment that is about $20 \times 20$ m in size. Using these five worlds, we set up six types of scenarios:

- **Lobby world with 34 pedestrians**: train our DRL control policies
- **Lobby world with 5-55 pedestrians**: test generalization across different crowd sizes
- **Autolab world with 25-35 pedestrians**: test generalization to unseen environments and crowd densities
- **Cumberland world with 25-35 pedestrians**: test generalization to unseen environments and crowd densities




(a) Lobby (b) Autolab

(c) Cumberland (d) Freiburg

(e) Square

Figure 5.3. Gazebo simulation environments. The Lobby world has two configurations with 34 pedestrians and 5-55 (sampling interval 10) pedestrians respectively. The Autolab world, Cumberland world, Freiburg world, and Square world only have one configuration with 25-35 (sampling interval 10) pedestrians.

- **Square world with 25-35 pedestrians**: test generalization to unseen environments and crowd densities

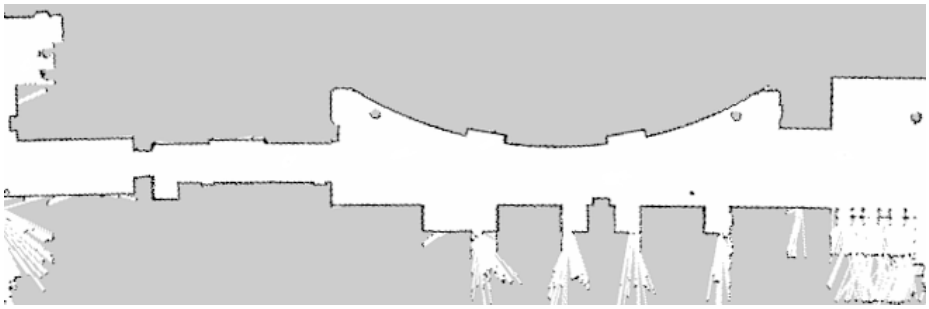
- **Freiburg world with 25-35 pedestrians**: test generalization to unseen environments and crowd densities

We use an Nvidia DGX-1 server with 40 CPU cores, 8x Tesla V100 GPUs with 16 GB memory, and 512 GB of RAM to train our DRL networks. We use a desktop computer with an Intel i7-6800K CPU, a GeForce GTX 1080 GPU with 8 GB memory, and 16 GB of RAM to run all the simulations. Both training server and desktop computer run Ubuntu 20.04, ROS Noetic Ninjemys, and Gazebo 11.5.1.

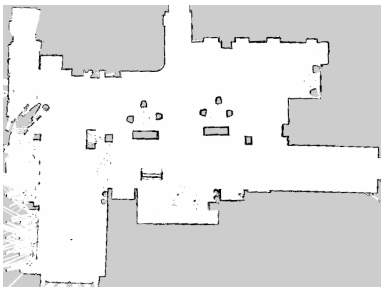
**Hardware Configuration.** In the hardware tests, the Turtlebot2 is equipped with an NVIDIA Jetson AVG Xavier embedded computer containing a cluster of 8 Carmel ARM cores, a 512 Core Volta GPU, and 16 GB RAM memory. The Xavier runs Ubuntu 18.04 system and ROS Melodic Morenia, and we use the 30 W power mode.

We test our robot in three environments, shown in Fig. 5.4. Figure 5.4(a) is the indoor hallway connecting the Engineering building and Science building, which is about $66 \times 5$ m in size. Figure 5.4(b) is the indoor structured lobby in the Engineering building, which is about $20 \times 10$ m in size. This is the same environment that we replicated in Gazebo to use for training our policy. Figure 5.4(c) is the outdoor hallway outside of the Science building, which is about $60 \times 8$ m in size.

We conduct two sets of hardware experiments. First, we test the real-world applicability in different crowd sizes using the indoor hallway environment (Fig. 5.4(a)). Second, we test the applicability in different environments using all three test environments.



(a) Indoor hallway



(b) Indoor lobby

(c) Outdoor hallway

Figure 5.4. Floorplans of real-world environments. All of these three environments are at Temple University.

### *5.3.2 Training Procedure*

We use the stable-baselines3 framework [129] to implement the PPO algorithm and train our DRL networks. We use the step decay with an initial learning rate of $10^{-3}$ as the learning rate schedule and set the mini-batch size to 512 during the training process. To train our control policies, we use the Lobby environment (Fig. 5.3(a)) with 34 pedestrians in it. The robot is then repeatedly assigned to reach a random goal from a random start position within the free space of the map. We used this procedure to train two different DRL-based control policies, one using the full reward (5.4) (DRL-VO) and one that does not use the heading direction reward $r_d^t$ (DRL).

### *5.3.3 Simulation Results*

We test these two DRL-based policies (*i.e.,* DRL and DRL-VO), along with other's DRL-based policies [71] (*i.e.,* A1-RD and A1-RC), the CNN-based policy [6], and the DWA planner [5], in each of the last five types of scenarios from Sec. 5.3.1. Note that we use these baseline policies directly from their papers [5, 6, 71] without any retraining or parameter tuning. We compare our algorithms to these baselines using four commonly used metrics from the navigation literature:

- **Success rate**: the fraction of collision-free trials
- **Average time**: the average travel time
- **Average length**: the average distance traveled
- **Average speed**: the average travel speed

We run four trials for each experimental configuration and control policy, where each trial consists of the robot navigating through a predefined series of 25 goal points around the corresponding test environment.

Note that we use the `amcl` ROS package to provide the robot localization service in known maps. All of these four trials use the same initial conditions, however, the results vary due to randomized sensor noise and variations in the pedestrian motion from the social force model.

**Different Crowd Densities.** We first focus on the Lobby environment with 5-55 (sampling interval 10) pedestrians used for testing the generalization ability across different

Table 5.2. Navigation results at different crowd densities.

| Environment | Method | Success Rate | Average Time (s) | Average Length (m) | Average Speed (m/s) |
|---|---|---|---|---|---|
| Lobby world, 5 pedestrians | DWA [5] | 0.93 | 12.10 | **5.08** | 0.42 |
| | CNN [6] | - | - | - | - |
| | A1-RD [71] | - | - | - | - |
| | A1-RC [71] | 0.91 | 14.11 | 6.14 | 0.44 |
| | DRL | 0.84 | 13.22 | 6.08 | 0.46 |
| | DRL-VO | **0.94** | **11.41** | 5.28 | **0.46** |
| Lobby world, 15 pedestrians | DWA [5] | 0.94 | 12.21 | **5.09** | 0.42 |
| | CNN [6] | - | - | - | - |
| | A1-RD [71] | 0.94 | 15.69 | 5.78 | 0.37 |
| | A1-RC [71] | 0.94 | 14.36 | 6.40 | 0.45 |
| | DRL | 0.80 | 13.66 | 6.24 | 0.46 |
| | DRL-VO | **0.95** | **11.34** | 5.26 | **0.46** |
| Lobby world, 25 pedestrians | DWA [5] | 0.82 | 13.49 | **5.12** | 0.38 |
| | CNN [6] | 0.80 | 19.31 | 6.16 | 0.32 |
| | A1-RD [71] | 0.90 | 17.87 | 6.07 | 0.34 |
| | A1-RC [71] | 0.88 | 14.18 | 6.26 | 0.44 |
| | DRL | 0.81 | 13.73 | 6.24 | 0.45 |
| | DRL-VO | **0.92** | **11.37** | 5.29 | **0.47** |
| Lobby world, 35 pedestrians | DWA [5] | 0.82 | 14.18 | **5.15** | 0.36 |
| | CNN [6] | 0.81 | 14.30 | 5.40 | 0.38 |
| | A1-RD [71] | 0.86 | 17.82 | 6.06 | 0.34 |
| | A1-RC [71] | 0.77 | 16.81 | 6.89 | 0.41 |
| | DRL | 0.75 | 14.30 | 6.46 | 0.45 |
| | DRL-VO | **0.88** | **11.42** | 5.31 | **0.46** |
| Lobby world, 45 pedestrians | DWA [5] | 0.77 | 15.39 | **5.16** | 0.34 |
| | CNN [6] | 0.79 | 16.65 | 5.62 | 0.34 |
| | A1-RD [71] | 0.76 | 23.16 | 6.61 | 0.29 |
| | A1-RC [71] | 0.77 | 14.65 | 6.28 | 0.43 |
| | DRL | 0.69 | 13.96 | 6.41 | 0.46 |
| | DRL-VO | **0.81** | **11.65** | 5.37 | **0.46** |
| Lobby world, 55 pedestrians | DWA [5] | 0.67 | 16.05 | **5.22** | 0.33 |
| | CNN [6] | 0.70 | 19.13 | 5.94 | 0.31 |
| | A1-RD [71] | 0.67 | 26.90 | 6.58 | 0.25 |
| | A1-RC [71] | 0.66 | 16.36 | 6.73 | 0.41 |
| | DRL | 0.60 | 13.39 | 6.13 | 0.46 |
| | DRL-VO | **0.79** | **11.76** | 5.39 | **0.46** |

crowd sizes. Table 5.2 presents the results obtained from these trials, where we can observe three key phenomena.

First, our DRL-VO policy has a much higher success rate than our DRL policy in each crowd size, while having a similar average speed. This shows that the proposed VO-based heading direction reward is beneficial and plays a key role in enabling the robot to maintain a good balance between collision avoidance and speed. Second, compared with other control policies, our DRL-VO policy has the highest success rate and the fastest average speed in every situation, with these differences growing as the density of pedestrians increases. This indicates that our policy is able to better generalize than the other approaches, especially compared to the CNN-based policy [6], which could not even reach the goal in some situations. Third, the average speed of the robot using our DRL-VO policy remains nearly constant across all situations, regardless of crowd density, allowing it to reach the goal most quickly. Compared to DWA [5], we see that our planner takes a slightly longer route to the goal, indicating a preference for giving a wider berth to pedestrians.

**Different Unseen Environments.** We next consider the navigation behavior of our DRL-VO policy in different unseen environments, namely the Autolab, Cumberland, Freiburg, and Square environments from Fig. 5.3. In each environment, we test two different crowd densities: 25 and 35 pedestrians. Tables 5.3 and 5.4 summarize these results. As can be seen, our novel DRL-VO policy still has both a higher success rate and slightly higher average speed than our DRL policy, with the only exception being a slightly lower speed in the Square world.

Table 5.3. Navigation results at different unseen environments with 25 pedestrians.

| Environment | Method | Success Rate | Average Time (s) | Average Length (m) | Average Speed (m/s) |
|---|---|---|---|---|---|
| Lobby world, 25 pedestrians | DWA [5] | 0.82 | 13.49 | **5.12** | 0.38 |
| | CNN [6] | 0.80 | 19.31 | 6.16 | 0.32 |
| | A1-RD [71] | 0.90 | 17.87 | 6.07 | 0.34 |
| | A1-RC [71] | 0.88 | 14.18 | 6.26 | 0.44 |
| | DRL | 0.81 | 13.73 | 6.24 | 0.45 |
| | DRL-VO | **0.92** | **11.37** | 5.29 | **0.47** |
| Autolab world, 25 pedestrians | DWA [5] | 0.88 | 13.50 | **5.47** | 0.41 |
| | CNN [6] | 0.78 | 21.59 | 6.61 | 0.31 |
| | A1-RD [71] | 0.88 | 18.92 | 6.54 | 0.35 |
| | A1-RC [71] | 0.77 | 15.63 | 7.35 | 0.43 |
| | DRL | 0.75 | 15.66 | 7.03 | 0.45 |
| | DRL-VO | **0.91** | **12.79** | 5.97 | **0.47** |
| Cumberland world, 25 pedestrians | DWA [5] | 0.78 | 15.16 | **5.53** | 0.37 |
| | CNN [6] | 0.63 | 27.80 | 7.73 | 0.28 |
| | A1-RD [71] | **0.91** | 17.47 | 6.63 | 0.38 |
| | A1-RC [71] | 0.81 | 16.31 | 6.93 | 0.43 |
| | DRL | 0.66 | 14.28 | 6.38 | 0.45 |
| | DRL-VO | 0.87 | **12.42** | 5.78 | **0.47** |
| Freiburg world, 25 pedestrians | DWA [5] | 0.83 | 13.90 | **5.75** | 0.41 |
| | CNN [6] | 0.48 | 30.38 | 8.30 | 0.27 |
| | A1-RD [71] | 0.81 | 17.95 | 6.61 | 0.37 |
| | A1-RC [71] | 0.74 | 17.06 | 7.35 | 0.43 |
| | DRL | 0.53 | 15.68 | 7.06 | 0.45 |
| | DRL-VO | **0.84** | **12.98** | 5.93 | **0.46** |
| Square world, 25 pedestrians | DWA [5] | 0.94 | 19.62 | **8.48** | 0.43 |
| | CNN [6] | 0.65 | 53.35 | 14.72 | 0.28 |
| | A1-RD [71] | **0.95** | 21.39 | 9.05 | 0.42 |
| | A1-RC [71] | 0.89 | 21.44 | 9.74 | 0.45 |
| | DRL | 0.83 | 22.62 | 10.77 | **0.48** |
| | DRL-VO | 0.86 | **18.19** | 8.61 | 0.47 |

This further demonstrates the utility of our proposed VO-based reward term to learn a more robust navigation control policy and achieve a better trade-off between collision avoidance and speed.

Table 5.4. Navigation results at different unseen environments with 35 pedestrians.

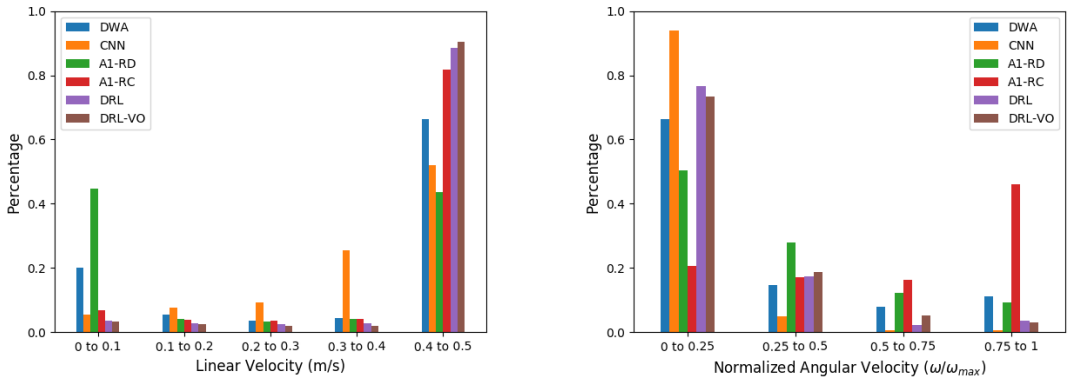
| Environment | Method | Success Rate | Average Time (s) | Average Length (m) | Average Speed (m/s) |
|---|---|---|---|---|---|
| Lobby world, 35 pedestrians | DWA [5] | 0.82 | 14.18 | **5.15** | 0.36 |
| | CNN [6] | 0.81 | 14.30 | 5.40 | 0.38 |
| | A1-RD [71] | 0.86 | 17.82 | 6.06 | 0.34 |
| | A1-RC [71] | 0.77 | 16.81 | 6.89 | 0.41 |
| | DRL | 0.75 | 14.30 | 6.46 | 0.45 |
| | DRL-VO | **0.88** | **11.42** | 5.31 | **0.46** |
| Autolab world, 35 pedestrians | DWA [5] | 0.81 | 16.52 | **5.55** | 0.34 |
| | CNN [6] | 0.68 | 22.73 | 6.65 | 0.29 |
| | A1-RD [71] | - | - | - | - |
| | A1-RC [71] | 0.73 | 17.11 | 6.93 | 0.41 |
| | DRL | 0.59 | 14.40 | 7.01 | 0.46 |
| | DRL-VO | **0.84** | **12.88** | 5.98 | **0.46** |
| Cumberland world, 35 pedestrians | DWA [5] | 0.74 | 16.28 | **5.57** | 0.34 |
| | CNN [6] | 0.60 | 24.25 | 7.08 | 0.29 |
| | A1-RD [71] | **0.88** | 18.04 | 6.69 | 0.37 |
| | A1-RC [71] | 0.77 | 15.37 | 6.66 | 0.43 |
| | DRL | 0.56 | 15.79 | 6.97 | 0.44 |
| | DRL-VO | 0.78 | **12.62** | 5.84 | **0.46** |
| Freiburg world, 35 pedestrians | DWA [5] | 0.70 | 18.15 | **5.78** | 0.32 |
| | CNN [6] | 0.57 | 20.09 | 6.66 | 0.33 |
| | A1-RD [71] | - | - | - | - |
| | A1-RC [71] | 0.65 | 17.11 | 6.93 | 0.41 |
| | DRL | 0.39 | 18.21 | 7.61 | 0.42 |
| | DRL-VO | **0.76** | **13.37** | 6.16 | **0.46** |
| Square world, 35 pedestrians | DWA [5] | 0.93 | 19.81 | **8.51** | 0.43 |
| | CNN [6] | 0.85 | 23.81 | 9.23 | 0.39 |
| | A1-RD [71] | **0.96** | 21.85 | 9.06 | 0.41 |
| | A1-RC [71] | 0.89 | 21.29 | 9.67 | 0.45 |
| | DRL | 0.86 | 21.48 | 10.27 | **0.48** |
| | DRL-VO | 0.87 | **18.32** | 8.66 | 0.47 |

Our DRL-VO policy also has the highest success rate out of all the algorithms in the Lobby, Autolab, and Freiburg worlds. Additionally, our policy yields a faster and more consistent speed than other policies and tends to have the shortest path, aside from the

DWA planner. This indicates that our proposed DRL-VO control policy has strong generalizability to different unseen environments.

The success rate of our policy drops, however, both in absolute and relative terms, in the Cumberland and Square worlds. Both of these environments have larger open spaces compared to the rest in Fig. 5.3. Given this, two potential causes of this decrease in success rate are: 1) the control policy was trained in the more structured lobby, so adding in more varied environments during training would further improve the generalizability, or 2) our DRL-VO policy with VO-based reward function is more suitable for structured environments rather than open spaces.

The A1-RD policy [71] has the highest success rate in the Cumberland and Square environments. We found this to be primarily due to its "stop and wait" strategy, where a robot that detects an obstacle in front of it will turn to a free space, stop, and wait for the obstacles to disappear. However, this behavior is the same regardless of the number of obstacles or whether these obstacles are static or dynamic. This can cause the robot to become frozen when it encounters a "C-shaped" obstacle because its policy only used raw lidar data and it was trained using a typical passive collision avoidance strategy based only on distance. This causes a robot using A1-RD to fail to reach the goal in both the Autolab and Freiburg environments with 35 pedestrians. In contrast to this, our DRL-VO policy can distinguish between static obstacles and dynamic pedestrians using the combination of lidar data and pedestrian kinematics. Moreover, depending on the crowd densities and velocity obstacle space, our DRL-VO policy will always tune its heading direction to actively avoid pedestrians and move towards the goal point. See the videos in Multimedia Extension 1 to see these differences in the behavior of A1-RD and DRL-VO in detail.




(a) Linear velocity distribution

(b) Angular velocity distribution

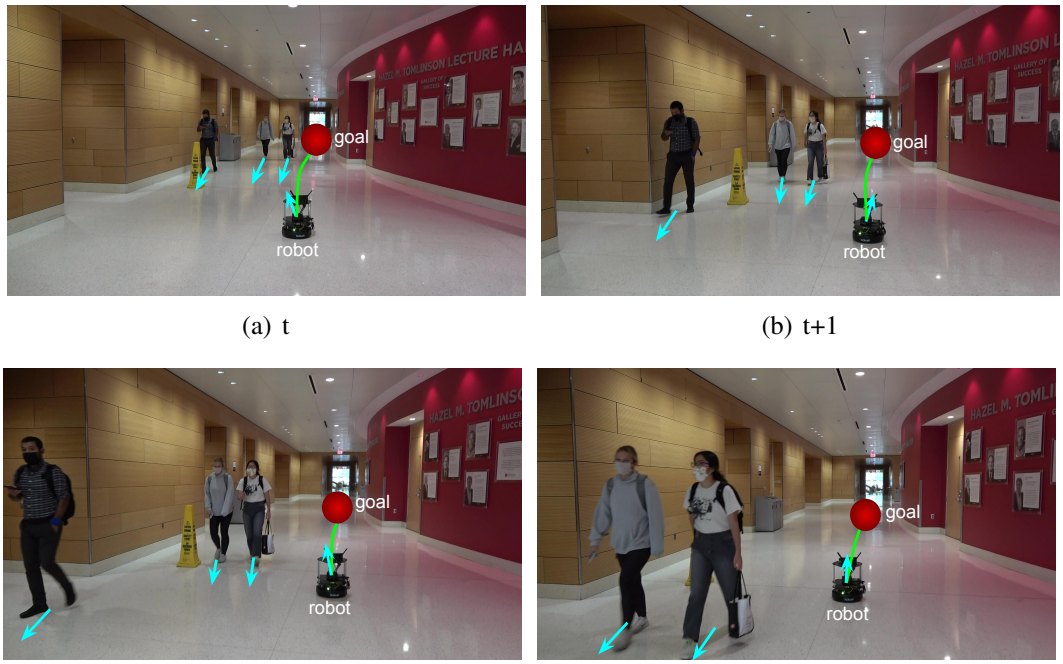
Figure 5.5. Histograms of the output velocities for different control policies. Data was collected from the lobby simulation world with 35 pedestrians.

**Output Velocity Distribution.** We also analyze the output velocity distribution of each tested control policy to better understand why our DRL-VO policy has a faster average speed than other model-based and learning-based control policies. To approximate the output velocity distribution, we divide the linear velocity from $[0, 0.5]$ m/s into 5 bins in 0.1 m/s increments, and the angular velocity from $[0, 1]$ $\omega_{\max}$ into 4 bins, and calculate the percentage of each bin. Note that we take the absolute value and normalize the angular velocity to ignore the direction of the turn and to account for different maximum values across the different control policies, respectively.

We observe a few interesting trends in the data, shown in Fig. 5.5. First, the majority of the linear velocities generated by all policies lie at either extreme of the range. In fact, our DRL and DRL-VO policies have the highest percentage of outputs in the range of $[0.4, 0.5]$ m/s (about 90%), which demonstrates why we achieve a consistently higher average speed. On the other hand, the A1-RD policy has the highest percentage in the range of $[0, 0.1]$ m/s (over 40%), showing the prevalence of the "stop and wait" strategy noted

above. The only other policy with a significant percentage of the output near 0 is DWA, which occurs due to deadlock in particularly crowded regions. The only policy to have any significant output in the middle of the range is our CNN policy, which was the only policy learned in a supervised setting. Second, most of the control policies have similar angular velocity distributions, with a majority of commands being small changes to update the heading angle of the robot (*i.e.,* between $[0, 0.25]$ of the maximum angular velocity). However, there are some key differences. First, the two outliers are the CNN policy, which yields very few commands outside of this minimum range, and the A1-RC policy, which generates a majority of commands at the upper range. This means that those policies generate motions with little turning or with a great deal of turning, respectively. Our DRL and DRL-VO policies generate fewer commands with high angular velocities, which is a result of the path smoothness term in our reward function, $r_w$.

**Failure Cases and Solutions.** Although our proposed DRL-VO policy is robust to different crowd densities and unseen environments, there are still some failure cases. One is that it can become difficult to find a collision-free direction in very high crowd densities. The other is the unexpected sudden changes in pedestrian movement, such as a person suddenly stepping out of a crowd. Our analysis of these failures is caused by three reasons: 1) our simulated pedestrians occasionally bump into one another due to the imperfect social force model, 2) there are occasionally missed pedestrian tracks in the MHT due to occlusion and ambiguous data association, and 3) the safety guarantees of VOs do not extend to the resulting learned policy because we cannot explicitly define collision constraints and apply them to the black box network. There are many possible ways to alleviate these issues in future work, including continuing to train the policies (learned in simulation) in real-world




(a) t (b) t+1

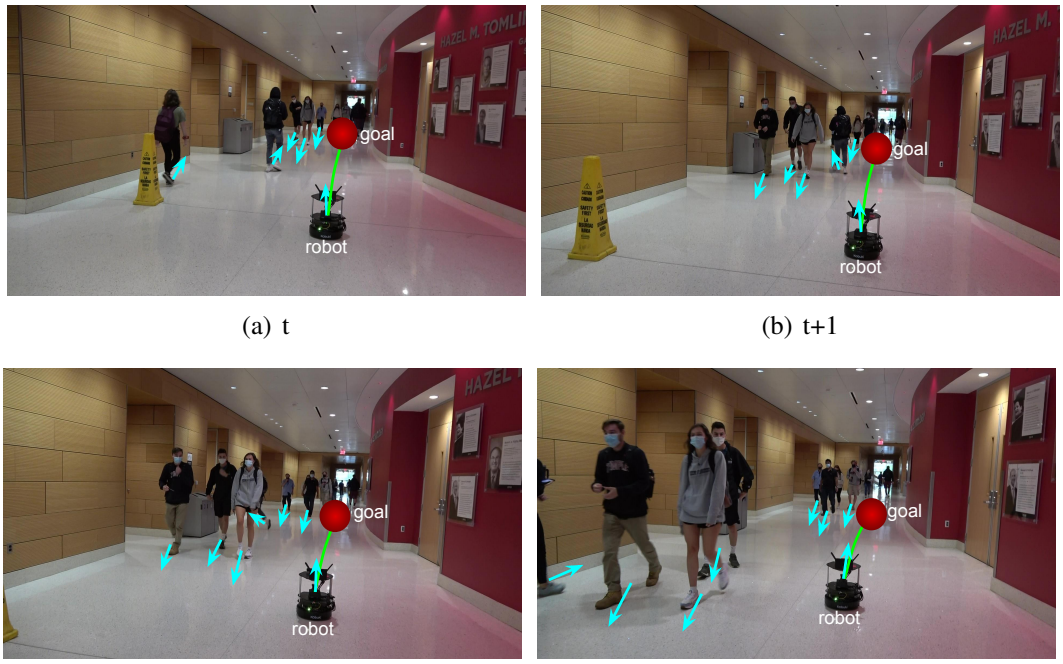
(c) t+2 (d) t+3

Figure 5.6. Robot reactions to moving pedestrians in the indoor hallway with the low crowd density at different times.

environments, improving the robustness of the MHT, adding probabilistic predictions of the future state of the environment, adding a control barrier function as a safety constraint like [130], or simply add a safety module as a backup: when the obstacle is too close to the robot, the safety module will override the learned policy to stop and rotate the robot until it finds a collision-free space (much like the "stop and wait" in the A1-RD policy).

### *5.3.4 Hardware Results*

Besides the simulated experiments, we also conduct a series of real-world experiments to demonstrate the applicability of our DRL-VO policy. Instead of performing hardware experiments in the laboratory under carefully controlled conditions as [71, 77, 80, 81, 110], we focus on robotic navigation in real and difficult environments by testing our system during the peak periods of pedestrian traffic in the intervals between classes. We test in




(a) t (b) t+1

(c) t+2 (d) t+3

Figure 5.7. Robot reactions to moving pedestrians in the indoor hallway with the medium crowd density at different times.

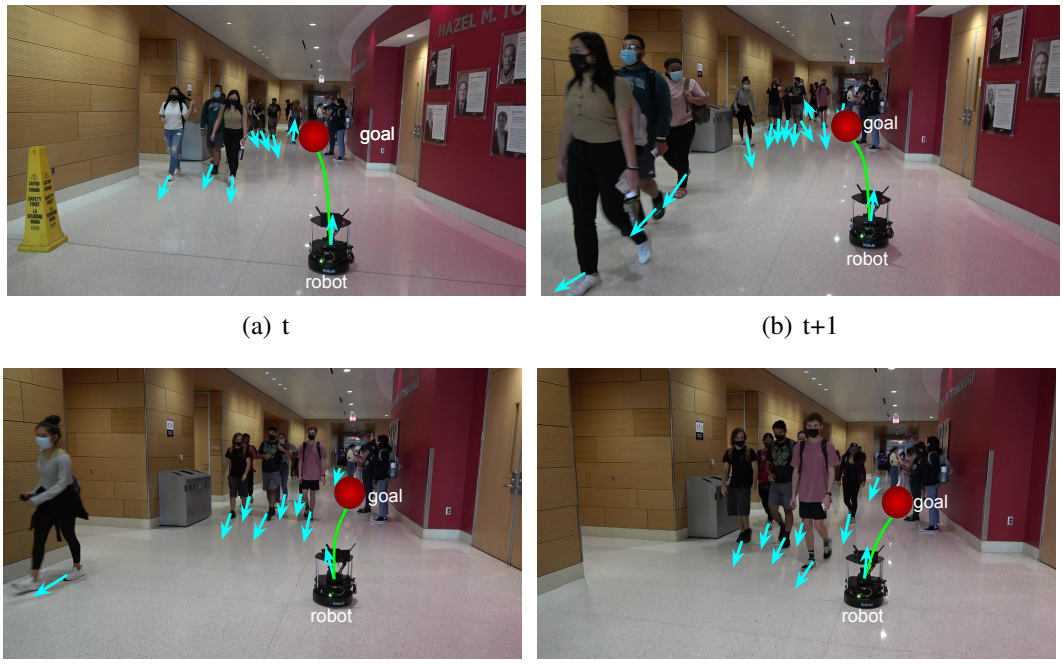
three different parts of campus, an indoor hallway, an indoor lobby, and an outdoor hallway environment, shown in Fig. 5.4. In each environment, the robot passes through a predefined series of goal points with students of different crowd sizes naturally walking around and causing the robot to adjust its path, as Fig. 5.6-5.10 shows.

The real Turtlebot2 robot directly uses our DRL-VO policy trained from our Gazebo simulation platform without any fine-tuning. The computational complexity of our DRL-VO policy is relatively low, with up to 60 FPS inference speed on the Jetson AVG Xavier embedded computer, far above the 40 Hz of the sensors. The space complexity of our DRL-VO policy is also low, with only 32.137 M parameters.This means our DRL-VO policy is clearly capable of real-time processing and is suitable for robots with limited resources.

**Different Crowd Densities.** To test the real-world generalization across different crowd sizes, we let the robot navigate through a sequence of 15 goal points in the indoor hallway shown in Fig. 5.4(a). We tested the robot at different times, getting results with different crowd densities. Figure 5.6-5.8 and Multimedia Extension 2 shows the schematic diagrams of robot navigation behavior and experimental videos for low, medium, and high crowd density.

In the low density crowd, Multimedia Extension 2 and Fig. 5.6 show how our robot safely drives a total of 131.24 m in the nearly empty hallway at an average speed of 0.41 m/s. It is apparent that in this case, our robot can easily navigate to its goals safely and quickly by tuning its heading direction towards sub-goal points and following the nominal path. Similarly, our robot is also able to safely and quickly navigate to its goal points through the medium-density crowds, traveling a total of 137.28 m at an average speed of 0.43 m/s without any collisions, as shown in Multimedia Extension 2 and Fig. 5.7. The high density crowd, shown in Multimedia Extension 2 and Fig. 5.8, is more complex than the first two cases as there are stationary pedestrians in addition to the groups of moving pedestrians. Even in such a complex dynamic environment, our robot is still able to actively avoid collisions with moving students going in or going out of classes, safely avoid static students standing or sitting at the hallway, and quickly reach predefined goals, traveling a total length of 133.40 m and an average speed of 0.41 m/s.

These results demonstrate that our proposed DRL-VO navigation policy is able to generalize to yet another new environment with varying crowd densities and, more importantly, is able to seamlessly transition from simulation to the real world without any




(a) t (b) t+1

(c) t+2 (d) t+3

Figure 5.8. Robot reactions to moving pedestrians in the indoor hallway with the high crowd density at different times.

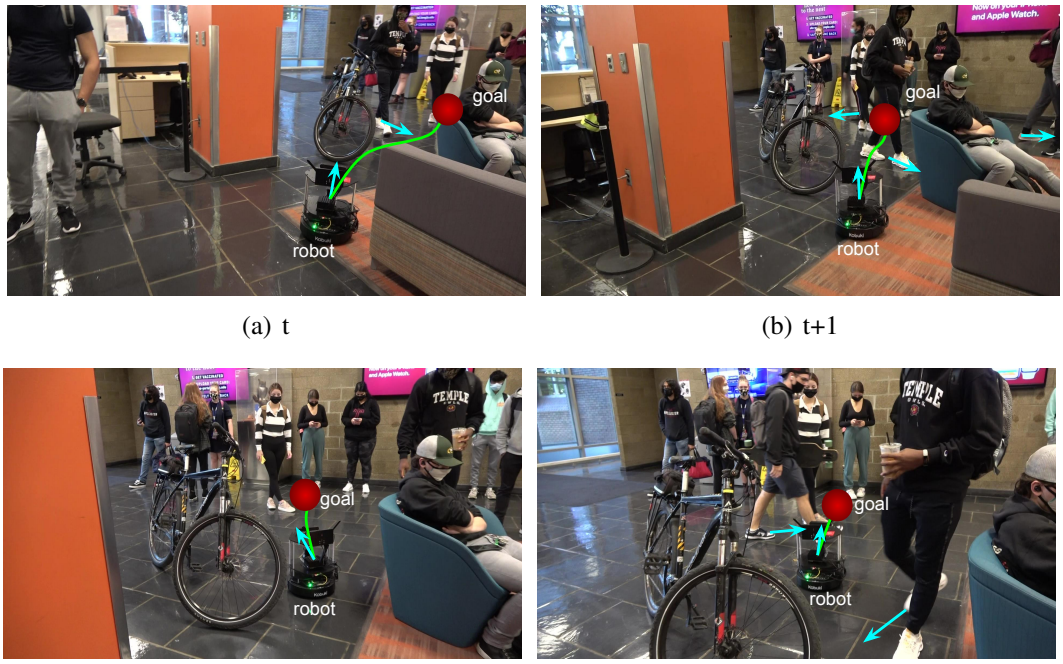
retraining or finetuning. Furthermore, we see that the trend of maintaining a high irrespective of the crowd density is consistent with the simulations in Sec. 5.3.3.

**Different Environments.** We also investigate the ability of our DRL-VO control policy to generalize across different real-world environments. Aside from the indoor hallway environment, we also let the robot navigate through different sequences of 10 goal points in both the indoor lobby and outdoor hallway environments, shown in Fig. 5.4(b) and Fig. 5.4(c), respectively. Note that all these three environments are tested under high crowd density, forming two sets of comparative experiments. One is the comparison of the structured indoor lobby and the relatively open indoor hallway. Another is the comparison of the indoor hallway and the outdoor hallway.

Multimedia Extension 3 video and Fig. 5.9 show the navigation behavior of our robot in the indoor lobby environment, where the robot successfully navigates through 10 goal points and travels a total of 68.96 m at an average speed of 0.37 m/s. Compared with the relatively open indoor hallway environment, this structured lobby environment is more complex and unpredictable, with static chairs, tables, bicycles, and trash bins as well as many dynamic pedestrians, resulting in a narrow and crowded free space. This structure yielded a more winding nominal path for the robot to follow, which frequently changed as pedestrians cut of different narrow passageways. Despite these complexities, we found no significant navigation differences between these two environments, and our robot was still able to safely and quickly reach its goals. This qualitative comparison demonstrates that our DRL-VO policy is able to generalize to relatively open environments and structured environments in the real world.

We next analyze robot navigation in the outdoor hallway environment. As can be seen from Multimedia Extension 3 and Fig. 5.10, our robot navigates smoothly in the dense crowds and goes to its predefined goals without any collisions. It travels a total of 102.73 m at an average speed of 0.43 m/s through this open space environment. Note that our outdoor hallway is more open than that indoor hallway, with no static obstacles but many more pedestrians walking around. Compared with navigating in the indoor hallway, the navigation behavior of our robot again shows no significant observable difference. This outdoor experiment indicates that our DRL-VO policy can also work reliably in outdoor environments and it is robust to both indoor and outdoor environments.

One interesting observation from these real-world experiments is the curiosity of the pedestrians, who frequently stopped to observe the robot's behavior, a phenomenon that we



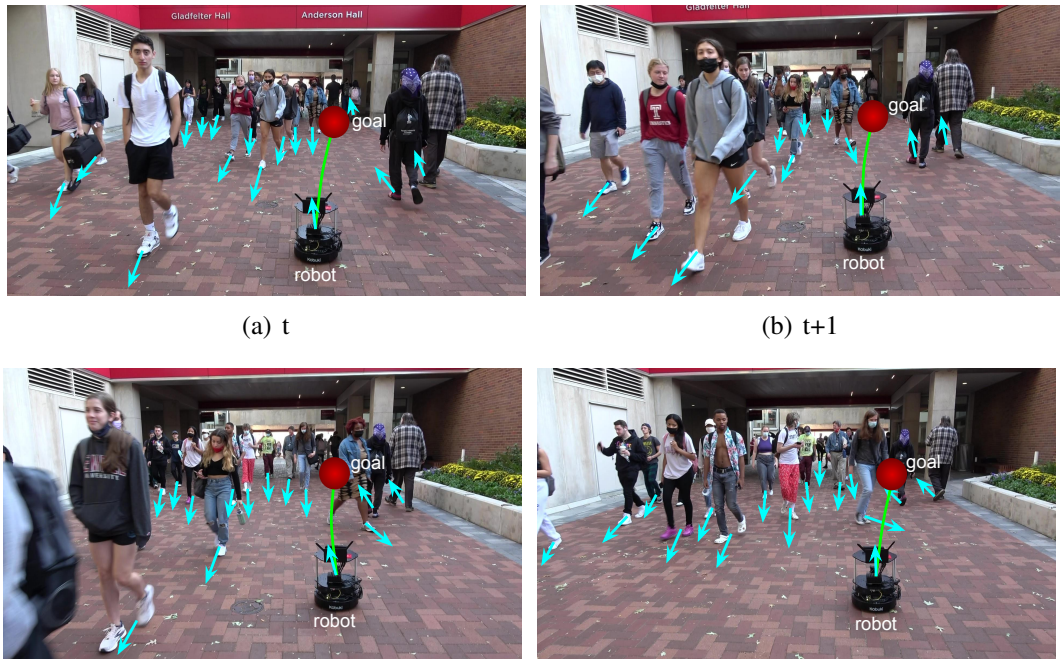

(a) t (b) t+1

(c) t+2 (d) t+3

Figure 5.9. Robot reactions to moving pedestrians in the indoor lobby with the high crowd density at different times.

did not include in the training data but which the robot was able to easily handle. We also observed that most people actively avoided the robot when they noticed it, giving it a wide berth. Recall we did model this behavior by adding an additional social force component to push people away from the robot and that this repulsion force was stronger than that between two pedestrians.

In summary, these real-world experiments and results demonstrate that our DRL-VO policy can smoothly transfer from simulation to reality and be applied to real-world tasks. We believe there are three main reasons for this. First, different from other works using 2D simulators [69, 71, 77–81, 83, 110, 112, 113], we use a 3D simulator to train our control policy, which shrinks the sim-to-real gap by modeling anthropomorphic 3D pedestrian and precise robot kinematics to give more realistic information than the point pedestrian and

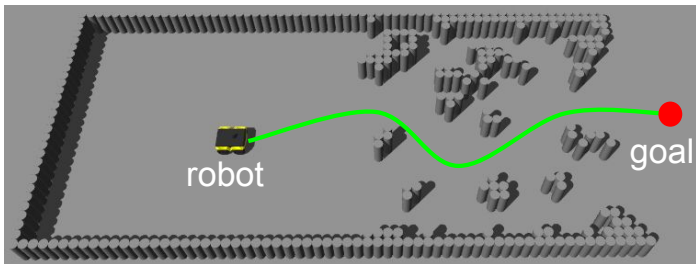



(a) t (b) t+1

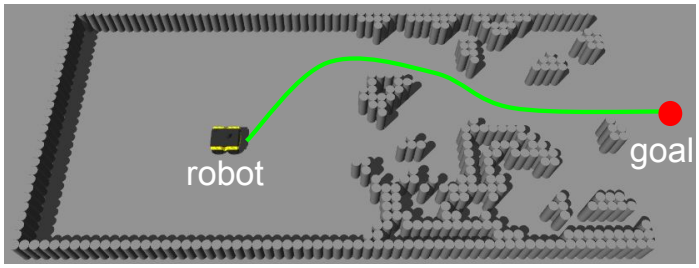
(c) t+2 (d) t+3

Figure 5.10. Robot reactions to moving pedestrians in the outdoor hallway with the high crowd density at different times.

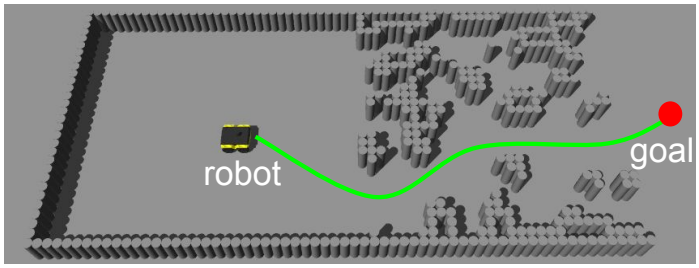
robot models. Second, we preprocessed the sensor data before inputting it into our navigation policy, turning the raw sensor data into the kinematic and lidar grids. We also were able to use the same software pipeline to process the data in both simulation and hardware. The resulting data structures are less sensitive to the differences between simulation and reality than raw data is (*e.g.,* simulated camera images are much different than real images due to the lack of detailed texture in the environment). Third, the VO-based heading direction reward function is a dense reward and always motivates the robot to actively move both towards goal points and away from potential collisions. All of these factors reduce the gap between simulation and reality, allowing our policy learned entirely in simulation to work in the real world without any retraining or fine-tuning.




(a) World 62




(b) World 120




(c) World 285

Figure 5.11. BARN simulation environment examples.

## 5.4 Highly Constrained Environments

Based on the excellent generalizability to different unseen crowded dynamic environments and crowd sizes, we further test our policy's ability to function in other tasks and on different robots with different sensors. To do this, we entered our DRL-VO control policy in the ICRA 2022 BARN Challenge [7], where the goal is to navigate a Clearpath Jackal robot through unknown and highly constrained static environments, such as those shown in Fig. 5.11. The competition consisted of two phases, simulation and hardware.

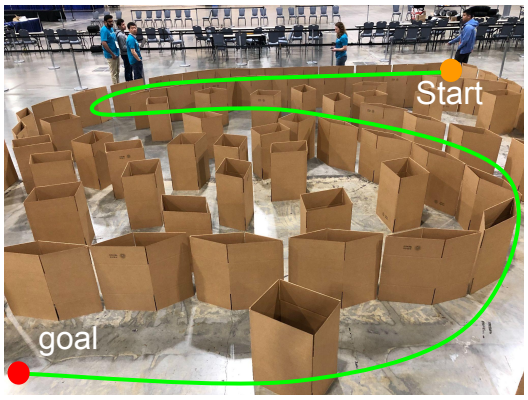
### *5.4.1 Experimental Setup*

**Robot Platform.** The BARN challenge uses a Jackal robot equipped with a Hokuyo UTM-10LX lidar. Compared to the Turtlebot2 platform used in Sec. 5.3, the maximum velocity of the Jackal is much greater (2 m/s vs. 0.5 m/s) and the sensing range and angular resolution of the UTM-10LX are both smaller (10 m and 0.375° per beam vs. 30 m and 0.25° per beam, respectively).

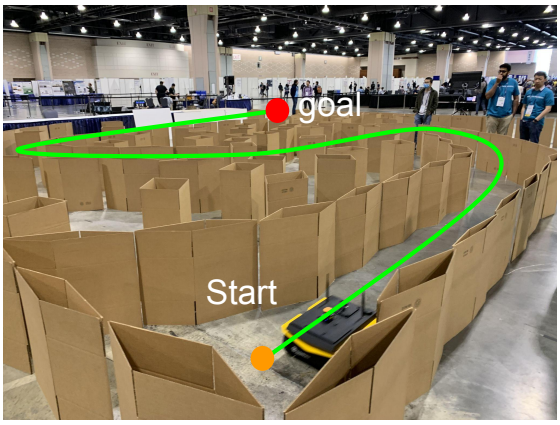
**Simulation.** The simulation portion of the challenge used 300 publicly available training Gazebo environments and 50 test environments known only to the competition managers. Each map is filled with cylindrical obstacles generated by the BARN environment generator [131], and the object density varies greatly across environments. The competition evaluated all policies on a desktop computer with an Intel Xeon Gold 6342 CPU @ 2.80 GHz (without GPU support).

**Hardware.** There were three different test environments, each about $10 \times 10$ m in size: wide "S", pointed "S", and narrow "S". Unlike the semi-closed BARN simulation environments, these three physical test environments are complex mazes with only one reasonable path, many sharp turns, gap traps, and narrow passages. Additionally, the obstacles in the physical environments at the Pennsylvania Convention Center were rhombuses, shown in Fig. 5.12. The physical Jackal robot was equipped with an onboard computer with an Intel i3-9100TE CPU and 16 GB RAM memory (with no GPU) running Ubuntu 18.04 and ROS Melodic Morenia.

(a) Wide "S" shaped




(b) Pointed "S" shaped




(c) Narrow "S" shaped

Figure 5.12. BARN real-world testing environments.

### *5.4.2 Deployment Adjustment*

We utilized the same DRL-VO policy from Sec. 5.3.2 (trained using the Turtlebot2) in the BARN Challenge. However, to successfully deploy policy on the Jackal robot and meet the requirements of the BARN challenge, we needed to adjust several settings of our overall system.

First, the BARN challenge maps are all unknown to the robot. Therefore, we removed the localization module (*i.e.,* `amcl` ROS package) and adapted our navigation system to operate using only odometry (*i.e.,* without a map).

Second, we need to adjust the input observation data $\mathbf{o}^t = [\mathbf{l}^t, \mathbf{p}^t, \mathbf{g}^t]$ of the DRL-VO network. For the lidar data $\mathbf{l}^t$, since the UTM-10LX has only 720 scan points instead of 1080 scan points, we use the full 270° FOV (instead of the 180° FOV for the Turtlebot2) to construct the $80 \times 80$ lidar historical map $\mathbf{l}^t$. We set the pedestrian data $\mathbf{p}^t = 0$ since the maps only contain stationary obstacles. Lastly, we use a sub-goal point with a look-ahead distance of 1 m instead of 2 m to generate the goal $\mathbf{g}^t$ as this allows the robots to better avoid obstacles in the highly constrained environments.

Finally, we found that directly mapping the normalized velocity from the control policy to the speed range of the Jackal would lead to very aggressive behavior. We hypothesize that this is an artifact of the Turtlebot2 being slower than many pedestrians so it nearly always moves at (close to) maximum speed. However, this caused the Jackal to frequently bump into obstacles. To solve this problem, we designed a simple maximum velocity switching mechanism:

$$v_{\max} = \begin{cases} 2\,\mathrm{m/s} & \text{if } d^t_{\text{obs}} \geq d_f \\ 0.5\,\mathrm{m/s} & \text{otherwise,} \end{cases} \tag{5.16}$$

where $v_{\max}$ is the maximum velocity limit of the robot, $d^t_{\text{obs}}$ is the distance to the obstacle closest to the robot at time $t$, and $d_f (= 2.2\,\mathrm{m})$ is a threshold value. Intuitively, this equation sets the maximum velocity to 2 m/s only when there are no obstacles in front of the robot and uses a slower velocity near obstacles (matching the training velocity).

### *5.4.3 Results*

We competed in both the simulation and hardware phases of the BARN competition.

**Simulation.** The BARN challenge simulation competition uses 50 unseen Gazebo environments to test navigation policies, running 10 trials for each environment. All trials use the same start and goal points, with the goal 10 m in front of the start, which allows the robot to plan a reasonable nominal path in the absence of an environmental map.

Policies were scored using a metric $s$ that considers the success rate, travel time, and environment difficulty [131]:

$$s_i = \mathbb{1}(E_i) \times \frac{T_i^o}{\text{clip}\left(T_i^a, 4T_i^o, 8T_i^o\right)}, \tag{5.17}$$

where $\mathbb{1}(E_i)$ is an indicator function that indicates whether the trial of environment $E_i$ is successful and collision-free, $\text{clip}(x,a,b) = \min(\max(x,a),b)$, $T_i^a$ is the actual travel time, and $T_i^o$ is the optimal travel time, which is defined as

$$T_i^o = \frac{L_p}{V_m}, \tag{5.18}$$

where $L_p$ is the length of the nominal path calculated by Dijkstra's algorithm, and $V_m$ is the maximum speed (*i.e.,* 2 m/s). Note that the optimal travel time indicates the environmental difficulty. The best navigation score is 0.25, which means that the robot can safely navigate to the goal point in $\leq 4T_i^o$.

A total of 11 policies (*e.g.,* model-based, learning-based, and hybrid) participated in the BARN challenge simulation competition, with the results shown in Table 5.5. Our DRL-VO policy has the average highest navigation score among all policies, and achieved

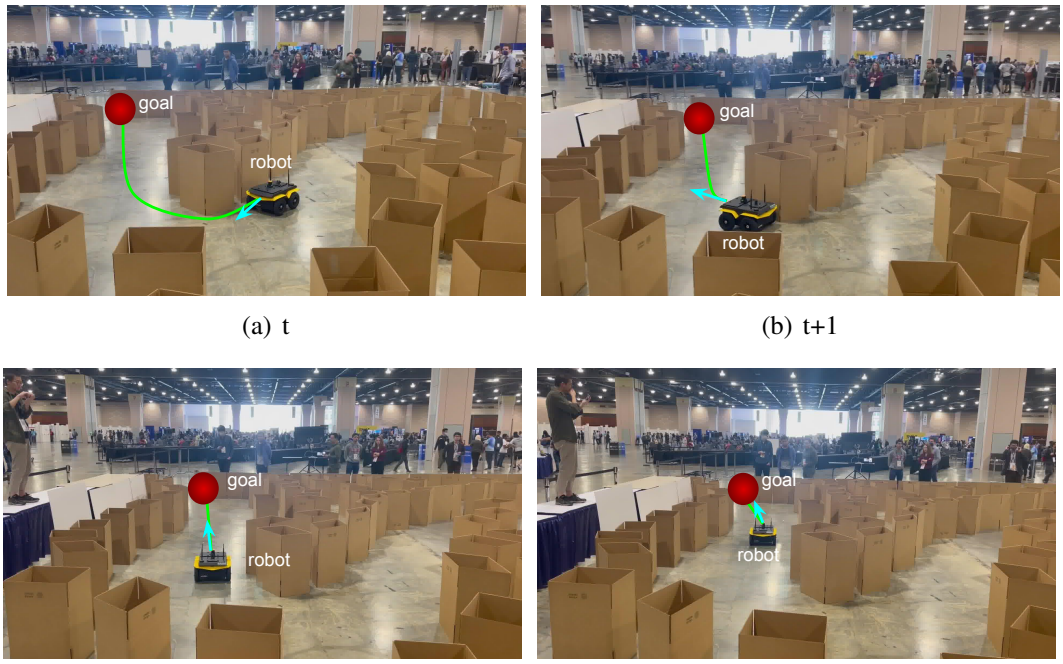
[2]`https://github.com/TempleRAIL/nav-competition-icra2022-drl-vo`
[3]`https://github.com/ShravanRavi2002/BARNSubmission`
[4]`https://github.com/nocholasrift/UVA-AMR-BARN--ICRA2022`

Table 5.5. Navigation results at BARN simulation environments.

| Method | Team | Average Score |
|---|---|---|
| **DRL-VO** [2] | **Temple TRAIL** | **0.2415** |
| LfLH [132] | Baseline | 0.2334 |
| AMRL [3] | UT AMRL | 0.2310 |
| DRAGON [4] | UVA AMR | 0.2200 |
| E-Band [133] | Baseline | 0.2053 |
| e2e [134] | Baseline | 0.2042 |
| APPLR-DWA [135] | Baseline | 0.1979 |
| Yiyuiii | NJU | 0.1969 |
| NavBot | IISc | 0.1733 |
| Fast DWA [114] | Baseline | 0.1709 |
| Default DWA [114] | Baseline | 0.1627 |

a score of 0.25 on all maps that it successfully completed (meaning the average speed was at least 0.5 m/s). See Multimedia Extension 4 to see our policy in these highly constrained environments. We did not receive full data about the test environments, but based on our experience in the training environments we saw that failures happened when the robot would just barely bump into an obstacle near very small gaps and sharp turns. We believe that this is because the Jackal has a larger physical size than the Turtlebot2, so the policy likely would have been safe with a Turtlebot2.

**Hardware.** Due to the excellent navigation performance achieved by our DRL-VO control policy in the BARN simulation competition, we were invited to participate in the physical competition (along with two other teams, ARML and DRAGON). The structure of the physical competition was to conduct three runs in each of the three test environments shown in Fig. 5.12. The team that completed the most runs (out of 9 total) won, with the



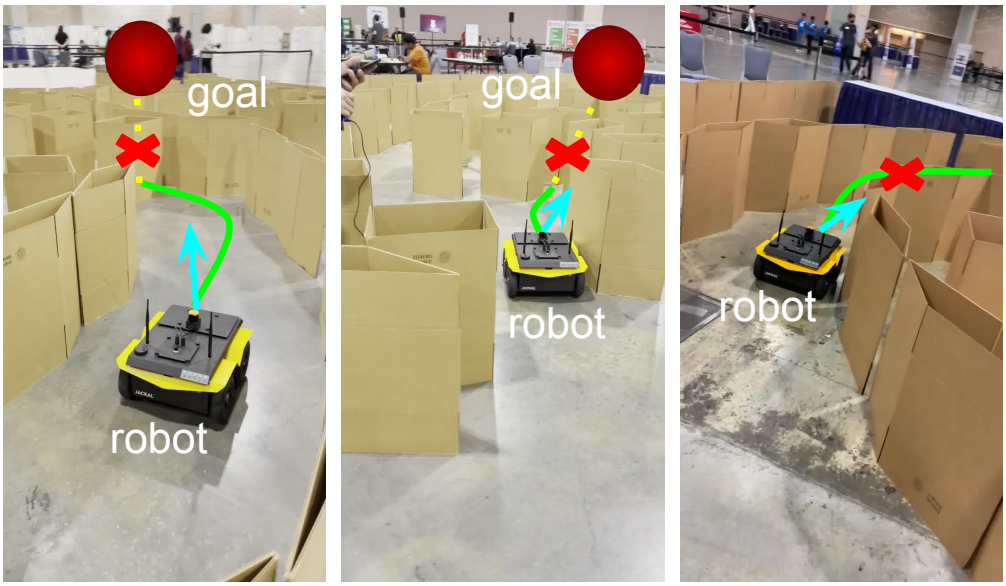

(a) t (b) t+1

(c) t+2 (d) t+3

Figure 5.13. Successful navigation behavior of the Jackal robot in the wide "S" shaped environment at different times.

average time used a tiebreaker. The test environments were generated by the competition managers that day and were not known to competitors beforehand. As we can see, these environments are more challenging than the simulation test environments, with many sharp turns and gap traps where the robot's field of view is highly occluded. Additionally, some of the passageways were so narrow that the Jackal robot could not turn around without hitting a wall.

We placed third in the physical competition, completing 2 out of 9 runs successfully (both in the first environment, Fig. 5.12(a)). Multimedia Extension 5 and Fig. 5.13 show one of these successful runs through the first test environment (Fig. 5.12(a)), which had relatively wide passages. However, we were not able to complete a full run in either of the other two environments due to two primary reasons: 1) there were no reasonable nominal




(a) Sharp turns (b) Gap traps (c) Narrow passages

Figure 5.14. Typical failure cases in the pointed/narrow "S" shaped environment. The yellow dotted lines are the unreasonable nominal paths given by the odometry-based global planner due to the limited FOV of the robot. The red crosses indicate potential collision areas.

paths to guide the robot through sharp turns or gap traps due to unpredictable obstacles limiting the robot's view, and 2) our DRL-VO policy is a bit aggressive and lacks precise control directives in the narrow passages, as it is trained in an environment without narrow passages and on a robot with a smaller physical size. See Multimedia Extension 5 and Fig. 5.14 for examples of these failures. In the future, we plan to alleviate these issues by using a map-based navigation system to give a better nominal global path to the robot and by training our DRL-VO policy in multiple different environments. Additionally, relative to the model-based approaches used by our competition, we could quickly retrain or fine-tune our learning-based model to work in each new environment.

### *5.4.4 Discussion*

Participating in the simulated and hardware BARN competition showed that our DRL-VO control policy, which was trained in a single crowded dynamic Lobby environment with 34 pedestrians using a Turtlebot2 robot, was able to generalize to a new navigation task (*i.e.,* highly constrained static environments) and work with different robot platforms and sensor configurations. We believe that this generalizability can be explained by the fact that we use preprocessed data representations (Sec. 5.2.1) rather than the raw sensor data. This creates a level of abstraction between the hardware/environment and the policy that allows it to be highly flexible. We will continue to explore the transferability of our policy in future work, particularly looking at how to account for changes in the physical properties of a robot. Two examples that we encountered in this work were the differences in the physical size of a robot (which we believe led to many of the collisions in the narrow passageways) and in the speed of the robot (since a Jackal robot moving in a crowd of people should vary its speed based on the people rather than going full speed).

## 5.5 Lessons Learned

Neural networks are, in essence, complicated tools to approximate functions. As such, they are particularly well suited to solve problems that are too complex for traditional rule-based methods, such as control policies to enable robots to navigate through dense crowds of pedestrians. There are three primary aspects to a neural network: the input data, the network structure, and the training methodology. In this section, we will discuss some of the broad lessons we have learned from our work in each of the above aspects.

### *5.5.1 Data Representations*

We argue that creating intermediate data representations (*i.e.,* hand-crafted features or learned features) from the raw sensor data allows us to learn navigation policies that better generalize to new scenarios than policies that directly use raw sensor data alone. As we conclude in Chapter 3, most robot navigation methods (*i.e.,* model-based [5, 59–61, 103–106] and learning-based [77–81, 83, 84, 110–112, 114]) utilize the preprocessed intermediate data as the input to their control policies, with most of the learning-based control policies also utilizing learned intermediate features. However, learning useful intermediate features from raw sensor data requires the use of photorealistic simulators and/or real data, both of which present significant challenges during the learning phase. Additionally, even photorealistic simulators differ (subtly) from reality, which can create a sim-to-real gap as the learned intermediate features depend on the data used to train the network.

Instead, we carefully handcrafted features in an attempt to concisely summarize the salient information needed for navigation while abstracting away the complexities of the real world. In our case, we choose two inputs to encode relevant data about the structure of the environment and the relative kinematics of pedestrians (or other moving objects). Specifically, we compress the lidar data by using minimum and average pooling to encode information about the basic structure of the space. This step allows our policy to avoid overfitting to the specific training environment, based on the findings of Pfeiffer et al. [122]. Similarly, we encode the RGB-D camera data by detecting objects from the RGB value, measuring positions using the depth data, and fusing these in the MHT to get information about pedestrian kinematics. Overall, these preprocessing steps form a compact and useful

representation that is far less sensitive to pixel-level differences between individual images than policies that use raw sensor data as input. This choice also affects the network design and training methodology, as we will discuss below.

### *5.5.2 Computational Constraints*

The goal of much machine learning research is simply to improve accuracy and/or reward. However, when using machine learning in the context of robotics it is critical to remember that the resulting algorithm will be deployed on a mobile robot with limited computational resources. Therefore, in practice, one must often make a good trade-off between accuracy/reward and computational efficiency. Based on our experience, we believe that having a short reaction time is more important than using a policy that is incrementally better (in the sense of lower training loss or higher reward in a simulator that runs slower than real-time) but that runs much slower and cannot be processed in real-time. In other words, we believe one should attempt to maximize reward subject to a constraint on processing time rather than the other way around. Future work will study this hypothesis in greater detail.

### *5.5.3 Training Methodology*

In a deep reinforcement learning setting, there are two key aspects to training: the environment and the reward design.

**Training Environment.** In general, reinforcement learning must take place in simulation due to the large number of iterations needed for the policy to converge and the need to explore potentially unsafe regions of the control space before convergence. One direction of work in the navigation research community is to develop photorealistic simu-

lators [136]. We agree that these are necessary to learn control policies that use raw images as their input. However, our input data structures provide a level of abstraction between the visual scene and the control policy by encoding the detail (or lack thereof) in the world in a consistent set of representations. This allows us to avoid using a computationally complex simulator and instead have an environment with flat visual textures and identical people-shaped rigid bodies that float along the ground. As a result, each iteration takes less time while still effectively learning a policy that generalizes well to new environments, including going from a simulated world to the real world without the need for any modifications.

Another important aspect of the training environment is the realism of human behavior. There are three types of pedestrian crowd models used by navigation-related researchers: social force-based model [6, 67, 71, 103], velocity obstacle-based model [72, 77–81, 83, 84, 110, 112], and pedestrian dataset-based model [108, 111, 113]. These works, especially those driven by pedestrian datasets, only modeled pedestrian dynamics and completely ignore any human-robot interactions.

However, during our real-world experiments, we observed three distinct types of human reactions to the robot, each of which affects the robot in a different way. The first, and most common, was for pedestrians to slightly adjust their path to avoid the robot but to continue on their way. Second, some pedestrians were curious about our robot and would often stop and watch or follow the robot to observe its behavior. Third, and least commonly, some pedestrians were afraid of the robot and would try to get away from the robot quickly. This list is by no means exhaustive, and there are likely many more human-robot interactions that affect navigation. Instead, this list is meant to demonstrate that approaches that completely ignore human reactions to robots will be less successful

and have a larger gap between simulation and reality. Our approach, also used by several other groups [67, 71], is a simplistic one: add an additional social force to repel people and robots away from one another. We believe that one fruitful direction for future work will be to explore the range of human behaviors, determine what effect they have on the robot, and develop methods to (approximately) reconstruct this behavior in a simulated setting.

**Reward Design.** The question of how to design an effective reward function for a specific application is still an open problem. For our robot navigation application, we want the prioritize collision avoidance while still reaching the goal in the shortest time possible. As we previously discussed, most DRL-based navigation policies use sparse reward functions, including a zeroth-order (*i.e.,* distance-based) collision avoidance term and a goal-attainment term, which guide the robot to avoid collisions and reach the goal, respectively. However, these zeroth-order collision avoidance terms force the robot to reactively avoid collisions instead of proactively taking evasive action. This was the main motivating factor in using velocity obstacles to create a first-order collision avoidance term, as the velocity information guides the robot to pick actions that will be safe both now *and in the future*. Our experiments on different crowd densities and unseen environments show that our novel first-order active reward function greatly improves the safety and speed of robot navigation compared to the zero-order passive reward function (DRL-VO vs. DRL in Tables 5.2–5.4). Based on these results, another avenue for future exploration will be designing other higher-order reward functions to improve robots' ability to proactively avoid collisions in highly dynamic scenes.

A secondary motivation for using velocity obstacles was to allow the robot to indirectly learn from successful model-based approaches. For example, controllers based

on velocity obstacles can guarantee collision avoidance under the assumption of perfect knowledge of the environment. However, in practice, that assumption is never valid. Therefore, instead of directly applying velocity obstacles to generate control actions, we use it as a reward term to guide behavior that is (at least close to) safe. Another avenue of future work will be to examine the effectiveness of using other model-based control approaches to guide the design of similar reward terms that combine safety and progress towards the goal into a single term.

## 5.6 Conclusion

In this chapter, we proposed a novel crowd-aware control policy (*i.e.,* DRL-VO) to enable autonomous robot navigation in crowded dynamic environments with both static obstacles and dynamic pedestrians, as well as in highly constrained static environments. From the detailed background material analysis, our DRL-VO control policy has two key novelties. First, we propose a new combination of preprocessed data representations, which can work well in crowded dynamic environments with up to 55 pedestrians and bridge the appearance gap between an imperfect simulation and reality. Specifically, the robot fuses a short history of lidar data, current pedestrian kinematics (*i.e.,* position and velocity stored in the two grids), and a sub-goal point. All of this data is represented within the robot's local coordinate frame, making our navigation policy robust to errors in localization that are common in crowded, dynamic environments.

Second, we design a novel velocity obstacle-based reward function used to train the policy, which can be easily applied to multi-robot navigation policies. This reward uses

first-order information about pedestrians to guide the robot to actively steer towards the goal while avoiding potential future collisions.

We demonstrate that our DRL-VO policy generalizes better and maintains a better balance between collision avoidance and speed to different crowd sizes and different unseen environments than other state-of-the-art model-based, supervised learning-based, and DRL-based policies, achieving an almost constant average speed but with a higher success rate over a series of 3D simulation experiments. We also demonstrate through extensive hardware experiments that our DRL-VO policy overcomes the sim-to-real gap and works reliably in real-world indoor and outdoor environments with different crowd sizes, and that it is able to do so without any fine-tuning or modification. Furthermore, by participating in the ICRA 2022 BARN simulation and real-world challenge competitions and placing 1st and 3rd respectively, we demonstrated that our DRL-VO policy can also generalize well to highly constrained environments, different input sensors, and different types of robots.

# CHAPTER 6
# ENVIRONMENTAL PREDICTION AND UNCERTAINTY-AWARE CONTROL POLICY

Many interesting works have focused on environment prediction problems, ranging from using traditional multi-object detection and tracking methods to deep learning (DL)-based methods. While traditional methods [86, 87] require multi-stage procedures and hand-designed features, most DL-based methods use an occupancy grid map (OGM) to represent the state of the perceived environment and transform the complex environment prediction problem into an OGM prediction problem, outlined in Fig 6.1. DL-based methods include image-based works [89–93], which treat the OGMs as images and use video prediction networks to solve it, and motion-based works [97–99, 102], which explore the ego-motion information and motion flow of the environment. However, image-based works only focus on improving network architectures and ignore much useful information from the kinematics and dynamics behind the environment, and most of these motion-based works are designed for vehicle scenarios where their network models require large memory and are time-consuming, which are not very suitable for mobile robots with limited resources. Furthermore, all the above-described works only provide deterministic OGM predictions and cannot estimate the uncertainty of future states, which is a key point in helping robots operate in complex, dynamic, and hazardous environments.

To address these issues, we propose a family of deep neural network-based Stochastic Cartographic Occupancy Prediction Engines (*i.e.,* SCOPE, SCOPE++, and SO-SCOPE) for resource-constrained mobile robots to provide stochastic future state predictions and enable

uncertainty-aware navigation in crowded dynamic scenes [1]. Specifically, we first exploit the kinematics and dynamics of the robot and its surrounding objects (*i.e.,* robot motion, object motion, and static scene geometry). Second, we use a variational autoencoder [137] (VAE)-based network to provide uncertainty estimates and predict a distribution over future states of the environment. Third, we perform software optimization on the performance of these predictors for real-time usage onboard mobile robots. Finally, we conduct simulated and real-world navigation experiments to demonstrate the performance of our SCOPE family in prediction and navigation.

## 6.1 Problem Formulation

We consider the problem of an unmanned ground vehicle (UGV) moving through an environment filled with both static infrastructure (*e.g.,* tables and walls) and dynamic objects (*e.g.,* people). As the robot moves, we assume that it is able to obtain accurate estimates of its own relative pose and velocity from its odometry sensors or other localization algorithms over short periods of time (on the order of 1 s). We denote the pose and velocity of the robot at time $t$ by $\mathbf{x}_t = [x_t \; y_t \; \theta_t]^T$ and $\mathbf{u}_t = [v_t \; w_t]^T$ respectively. We assume the robot is equipped with a 2D lidar sensor. Let $\mathbf{y}_t = [r_t \; b_t]^T$ denote the lidar measurements (range $r$ and bearing $b$) at time $t$.

We assume the world is 2.5D, as is commonly done in mobile robotics applications. Thus, we can represent the environment using a 2D occupancy grid map (OGM). Let $\mathbf{o}_t$ de-

[1]We have open-sourced the OGM prediction code with the OGM dataset (`https://github.com/TempleRAIL/scope` and `https://github.com/TempleRAIL/SOGMP`) and its predictive uncertainty-aware navigation framework (`https://github.com/TempleRAIL/scope_nav`). Multimedia is also available at `https://youtu.be/xJBtWQDLU04`.

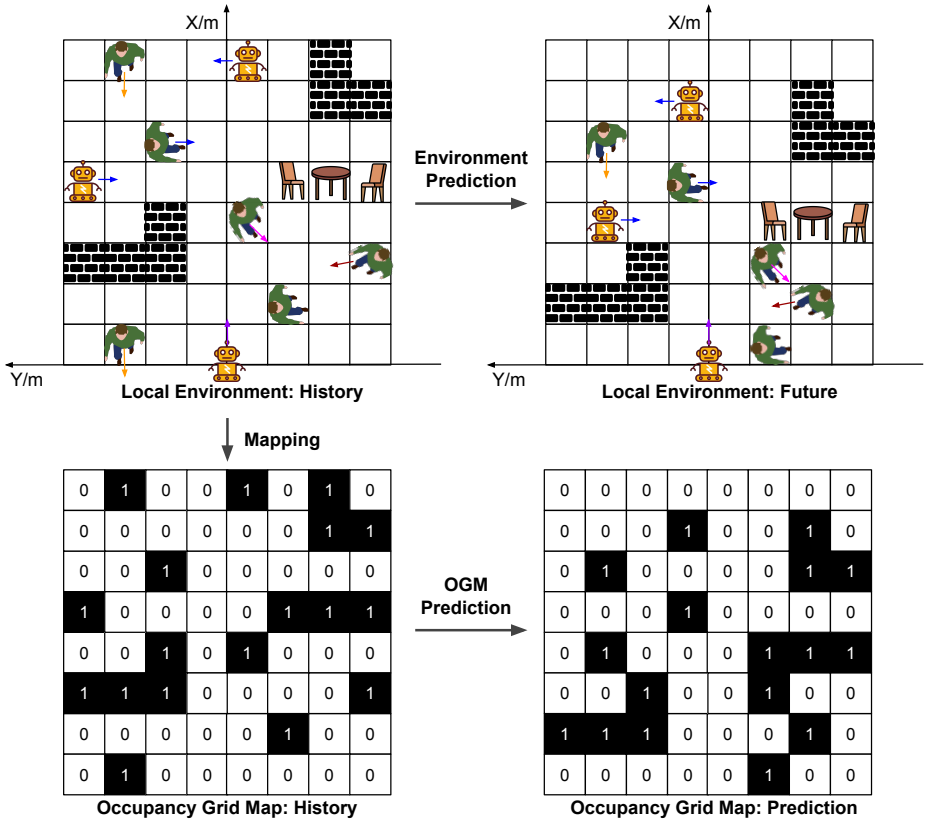



Figure 6.1. A simple illustration of the OGM prediction problem. In a complex dynamic environment with many pedestrians, robots, tables, chairs, and walls, colored arrows indicate the velocity of each agent.

note the OGM at time $t$. We will consider OGMs that are both binary (*i.e.,* each cell is either occupied or free) and probabilistic (*i.e.,* each cell has a probability of being occupied).

Given a history of $\tau$ lidar measurements $\mathbf{y}_{t-\tau:t}$ and robot states $\{\mathbf{x}_{t-\tau:t}, \mathbf{u}_{t-\tau:t}\}$, we consider the problem of predicting the future state (*i.e.,* OGM) of the environment $\mathbf{o}_{t+1}$. For notational compactness, let $\mathbf{d}_{t-\tau:t} = \{\mathbf{x}_{t-\tau:t}, \mathbf{u}_{t-\tau:t}, \mathbf{y}_{t-\tau:t}\}$ denote the history of data over the given window. We can formulate this as a prediction model:

$$p_\theta(\mathbf{o}_{t+1} \mid \mathbf{d}_{t-\tau:t}), \tag{6.1}$$

where $\theta$ are the model parameters, and the goal is to find the optimal $\theta^*$ to maximize (6.1). Note that this model can also be used in a rollout to predict multiple time steps into the future, *i.e.,* using $t+1$ to predict $t+2$, etc.

Note that in this paper, we set the lidar history to $\tau = 10$, the sampling rate is 10 Hz, and limit the physical size of OGMs to $[0, 6.4]$ m along the x axis (forward) and $[-3.2, 3.2]$ m along the y axis (left) in the robot's local coordinate frame. We use a cell size of 0.1 m, resulting in $64 \times 64$ OGMs. These settings are consistent with other works on mobile robot navigation [138]. All data $\mathbf{o}, \mathbf{u}, \mathbf{y}$ are represented in the *local* coordinate frame of the robot.

### *6.1.1 Image-Based Prediction*

An OGM can be considered a grayscale image, with the probability of occupancy defining the "color" in each cell of the regular grid. Using this interpretation, one could use image/video prediction algorithms to generate the next OGM given a short sequence of past OGMs. The prediction model (6.1) of image-based approaches is rewritten as

$$\mathbf{o}^*_{t+1} = \arg\max\, p_\theta(\mathbf{o}_{t+1} \mid \mathbf{d}_{t-\tau:t}) = f_\theta(\mathbf{o}_{t-\tau:t}), \tag{6.2a}$$

$$\mathbf{o}_{t-\tau:t} = e(\mathbf{y}_{t-\tau:t}), \tag{6.2b}$$

where $f_\theta(\cdot)$ is a deep neural network model, and $e(\cdot)$ is the conversion function to convert the lidar measurements to the binary OGMs in the robot's local coordinate frame (*i.e.,* using ray tracing). From this image-based model (6.2), we can easily see that image-based methods explicitly ignore the kinematics and dynamics of the robot and surrounding objects

(*i.e.,* they assume motion can be implicitly captured by very powerful network architectures and enough good data), and fail to provide a range of possible and reliable OGM predictions (*i.e.,* they assume a deterministic future).

### *6.1.2 SCOPE++*

Based on the limitations of image-based methods above, we argue that: 1) the future state of the environment explicitly depends on the motion of the robot itself, the motion of dynamic objects, and the state of static objects within the environment; 2) the future state of the environment is stochastic and unknown, and a range of possible future states helps provide robust predictions. With these two assumptions, we fully and explicitly exploit the kinematic and dynamic information of these three different types of objects, utilize the VAE-based network to provide stochastic predictions, and finally propose our novel stochastic OGM predictor SCOPE (shown in Fig. 6.2) to predict the future state of the environment. We also create a second variant called SCOPE++ that explicitly separates out static objects. Since SCOPE++ is a superset of SCOPE, the remainder of this section will focus on SCOPE++.

The prediction model (6.1) of SCOPE++, outlined in Fig. 6.2, can be rewritten as

$$\mathbf{o}_{t+1} \sim p_\theta(\mathbf{o}_{t+1} \mid \mathbf{d}_{t-\tau:t}) = p_\theta(\mathbf{o}_{t+1} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m}), \tag{6.3a}$$

$$\hat{\mathbf{o}}_{t+1} = h(\mathbf{o}_{t-\tau:t}), \tag{6.3b}$$

$$\mathbf{o}_{t-\tau:t} = e(\mathbf{y}^R_{t-\tau:t}), \tag{6.3c}$$

$$\mathbf{m} = g(\mathbf{y}^R_{t-\tau:t}), \tag{6.3d}$$

$$\mathbf{y}^R_{t-\tau:t} = k(\mathbf{d}_{t-\tau:t}). \tag{6.3e}$$

In this framework, (6.3a) is the VAE predictor module from which we sample future maps $\mathbf{o}_{t+1}$. (6.3b) and (6.3c) represent the dynamic object module, where $h(\cdot)$ processes the time series data for dynamic objects, $e(\cdot)$ is the conversion function like (6.2b), and $R$ denotes the local coordinate frame of the robot at predicted time $t+n$. (6.3d) is the static objects module, where $g(\cdot)$ is the occupancy grid mapping function for static objects. Finally, (6.3e) is the robot motion module, where $k(\cdot)$ is the transformation function to compensate for robot motion.

Note that (6.3) only predicts future states at the next time step $t+1$ and is used for training. To predict a multi-step future state at time $t+n$, we can easily utilize the autoregressive mechanism and feed the next-step prediction back $\mathbf{o}_{t+1}$ to our SCOPE/SCOPE++ network (6.3a) for $n-1$ time steps to predict the future states at time step $t+n$. Note that the prediction horizon $n$ could theoretically be any time step.

### *6.1.3 Robot Motion Compensation* $k(\cdot)$

To account for the robot's motion in dynamic scenarios, we propose a simple and effective ego-motion compensation mechanism $k(\cdot)$ to mitigate its dynamic effects and allow the environment dynamics to be consistent in the robot's local coordinate frame. To predict the future OGM state in the robot's local future view, we consider the robot's future ego-motion and transform the observed OGMs $\mathbf{o}_{t-\tau:t}$ to the robot's local coordinate frame at prediction time step $t+n$. So, our proposed ego-motion compensation mechanism can be divided into two steps: robot pose prediction and coordinate transformation.

**Robot Pose Prediction.** To account for the robot's future ego-motion, we need to predict the future pose of the robot at prediction time step $t+n$. Since the constant ve-

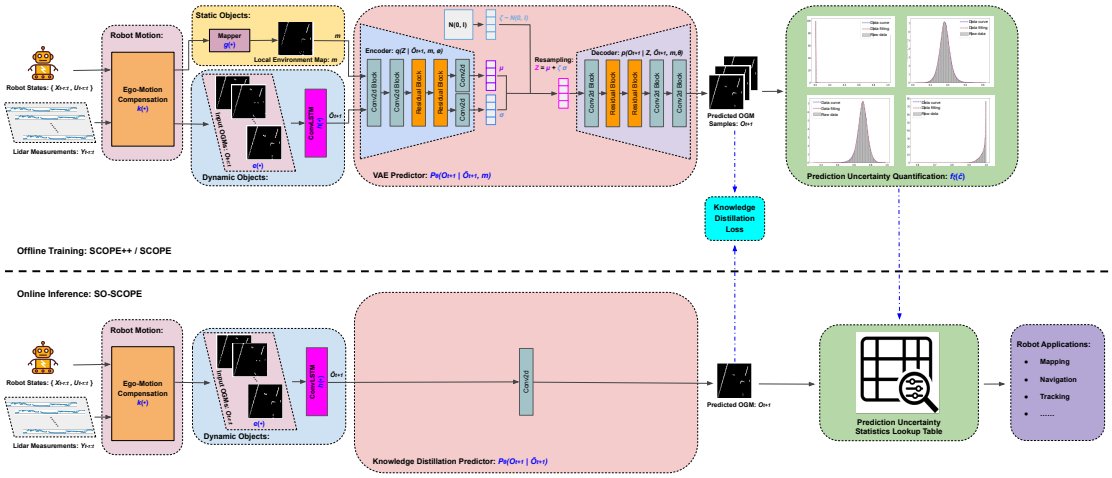



Figure 6.2. System architectures of the proposed SCOPE++ predictor, SCOPE predictor, and its software-accelerated SO-SCOPE predictor (note that SCOPE omits the Static Objects block compared to SCOPE++). The basic process of the SCOPE++ predictor is: 1) based on a history of robot states, the robot transfers the lidar measurement history to the predicted coordinate frame of the robot to compensate for the ego-motion, 2) these compensated lidar measurements are used to generate a local environment map to account for static objects, and a set of OGMs to account for dynamic objects, and 3) the local map of static objects and the predicted OGM of dynamic objects are fed into an variational autoencoder to predict the future OGM. To accelerate the SCOPE++ predictor, we first follow the SCOPE network architecture and replace the VAE network with a single convolutional layer, then use knowledge distillation technology to train the SO-SCOPE network to obtain the prediction information, and finally, we model and quantify the prediction uncertainty of the SCOPE++ to obtain uncertainty statistics and use them to generate uncertainty estimates of SO-SCOPE.

locity motion model is the most widely used motion model for tracking [139] and often outperforms more state-of-the-art methods in general settings [121], we use it as our robot motion model and assume that the robot keeps constant motion in a relatively short period (*i.e.,* less than 1 second). Note that other more suitable robot motion models specific to particular robot models can be used to provide better robot pose predictions. Then, we can

easily predict the future pose of the robot $\mathbf{x}_{t+n}$ at prediction time step $t+n$ using the robot's current pose $\mathbf{x}_t$ and velocity $\mathbf{u}_t$:

$$\begin{bmatrix} x_{t+n} \\ y_{t+n} \\ \theta_{t+n} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ w_t \end{bmatrix} n\Delta t + \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_\theta \end{bmatrix}, \tag{6.4}$$

where $\Delta t$ is the sampling interval, $\sigma_{(\cdot)}$ is the Gaussian noise.

**Coordinate Transformation.** To compensate for the ego-motion of the robot, we first use a homogeneous transformation matrix to transform the robot poses $\mathbf{x}_{t-\tau:t}$ to the robot's local future coordinate frame $R$:

$$\begin{bmatrix} x^R_{t-\tau:t} \\ y^R_{t-\tau:t} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_{t+n}) & -\sin(\theta_{t+n}) & x_{t+n} \\ \sin(\theta_{t+n}) & \cos(\theta_{t+n}) & y_{t+n} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{t-\tau:t} \\ y_{t-\tau:t} \\ 1 \end{bmatrix}, \tag{6.5a}$$

$$\theta^R_{t-\tau:t} = \theta_{t-\tau:t} - \theta_{t+n}. \tag{6.5b}$$

Then, by adding these ego-motion displacements, we convert the observed lidar measurements $\mathbf{y}_{t-\tau:t}$ from Polar to Cartesian coordinates:

$$\begin{aligned} \mathbf{y}^R_{t-\tau:t} &= \begin{bmatrix} x^z_{t-\tau:t} \\ y^z_{t-\tau:t} \end{bmatrix} \\ &= \begin{bmatrix} x^R_{t-\tau:t} \\ y^R_{t-\tau:t} \end{bmatrix} + r_{t-\tau:t} \begin{bmatrix} \cos(b_{t-\tau:t} + \theta^R_{t-\tau:t}) \\ \sin(b_{t-\tau:t} + \theta^R_{t-\tau:t}) \end{bmatrix}. \end{aligned} \tag{6.6}$$

Finally, we implement the transformation function (6.3e) and obtain a set of observed lidar measurements $\mathbf{y}^R_{t-\tau:t}$ at the robot's local future coordinate frame $R$. The benefit of ego-motion compensation is that we can treat these lidar measurements from a moving lidar sensor as observations in common world-fixed frame $R$, which significantly reduces the difficulty of OGM predictions and improves accuracy. Note this only requires the robot to have accurate odometry over the history window $\tau$ (typically on the order of 1 s), but not accurate global localization.

### *6.1.4 Dynamic Object Prediction* $h(\cdot)$

Since we use the OGM to represent the environmental state, we first need to implement a conversion function $e(\cdot)$ to convert the compensated lidar measurements $\mathbf{y}^R_{t-\tau:t}$ to the corresponding OGMs $\mathbf{o}_{t-\tau:t}$ by using coordinate to subscript conversion equations, which is shown in Algorithm 6.1.

Tracking and predicting dynamic objects such as pedestrians is the hardest part of environmental prediction in complex dynamic scenes. It requires some techniques to process a set of time series data to capture the motion information. While the traditional particle-based methods [87] require explicitly tracking objects and treat each grid cell as an independent state, recent learning-based methods [89–93, 97–99, 102] prefer to use RNNs to directly process the observed time series OGMs. Based on these trends, we choose the most popular ConvLSTM unit to process the spatiotemporal OGM sequences $\mathbf{o}_{t-\tau:t}$. However, while other works [90, 96] explicitly decouple the dynamic and static/unknown objects and use different networks to process them separately, we argue that the motion of dynamic objects is related to their surroundings, and that explicit disentangling may lose

**Algorithm 6.1:** Converting lidar points to OGMs

**Input:** compensated lidar measurements $\mathbf{y}^R_{t-\tau:t}$

**Input:** grid cell size $s$

**Input:** the physical size of the OGMs $S$

**Input:** the lower left corner of the OGMs $(x_0, y_0)$

**Output:** OGMs $\mathbf{o}_{t-\tau:t}$

```
initialize: o_{t-τ:t} = 0
for all parallel beams z ∈ y^R_{t-τ:t} do
    i = ⌊(x^z_{t-τ:t} − x_0)/s⌋
    j = ⌊(y^z_{t-τ:t} − y_0)/s⌋
    if i, j ∈ [0, S/s] then
        o_{t-τ:t}(i, j) = 1
    end if
end for
```

some useful contextual information. For example, pedestrians walking through a narrow corridor are less likely to collide with or pass through surrounding walls. To exploit the useful contextual information between dynamic objects and their surrounding, we directly feed the observed OGMs $\mathbf{o}_{t-\tau:t}$ into a ConvLSTM unit $h(\cdot)$ and implicitly predict the future state $\hat{\mathbf{o}}_{t+1}$ of dynamic (and static) objects.

### *6.1.5 Static Object Segmentation* $g(\cdot)$

While predicting dynamic objects plays a key role in environmental prediction, paying extra attention to static objects is also important to improve prediction accuracy. The main reason is that the area occupied by static objects is much larger than that of dynamic objects, as shown in Fig. 6.2, where static objects such as walls are consistently clustered together, while dynamic objects such as pedestrians are sparse and scattered point clusters.

**Algorithm 6.2:** GPU-accelerated OGM mapping

**Input:** compensated lidar measurements $\mathbf{y}^R_{t-\tau:t}$

**Output:** local environment map $\mathbf{m}$

1: **for all** time steps $n$ from $t-\tau$ to $t$ **do**

2: **for all parallel** grid cells $\mathbf{m}_i$ in the perceptual field of $\mathbf{y}^R_n$ **do**

3: $l_i = l_i + \log \frac{p(\mathbf{m}_i|\mathbf{y}^R_n)}{1-p(\mathbf{m}_i|\mathbf{y}^R_n)} - \log \frac{p(\mathbf{m}_i)}{1-p(\mathbf{m}_i)}$

4: **end for**

5: **end for**

Another reason is that static objects contribute to the scene geometry and give a global view of the surroundings, where these static objects maintain their shape and position over time.

To account for static objects, we utilize a local static environment map $\mathbf{m}$ as a prediction for future static objects, which is a key contribution of our work. We generate this local environment map $\mathbf{m}$ using the standard inverse sensor model [140]. This Bayesian approach generates a robust estimate of the local map, where dynamic objects are treated as noise data and removed over time. We speed up this step by implementing a GPU-accelerated OGM mapping algorithm[2] $g(\cdot)$ that parallelizes the independent cell state update operations by taking advantage of the additive update rule, as shown in Algorithm 6.2.

### *6.1.6 VAE Predictor*

The dynamic and static prediction functions from (6.3b) and (6.3d) yield one estimate of the future environment. However, it is unlikely that this exact map will be correct. To account for this, we wish to generate a range of possible future environment states that capture the uncertainty of the environment. We do this using a variational autoencoder

[2] `https://github.com/TempleRAIL/occupancy_grid_mapping_torch`

(VAE), a generative machine learning technique that creates samples that are "similar" to the predicted OGM $\hat{\mathbf{o}}_{t+1}$.

To represent the stochasticity of environment states, we assume that environment states $\mathbf{o}_{t-\tau:t+n}$ are generated by some unobserved, random, latent variables $\mathbf{z}$ that follow a prior distribution $\mathbf{z} \sim p_\theta(\mathbf{z})$. Then, our VAE predictor model (6.3a) can be rewritten as

$$p_\theta(\mathbf{o}_{t+1} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{o}_{t+1} \mid \mathbf{z}, \hat{\mathbf{o}}_{t+1}, \mathbf{m}) dz. \tag{6.7}$$

However, we are unable to directly optimize this marginal likelihood and obtain optimal model parameters $\theta$ because the true posterior $p_\theta(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})$ is unknown and intractable. Instead, we define a flexible variational approximation $q_\phi(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})$ to approximate the true posterior $p_\theta(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})$.

Then, we use a VAE network to parameterize our prediction model $p_\theta(\mathbf{o}_{t+1} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})$, outlined in Fig. 6.2, where the inference network (encoder) parameterized by $\phi$ refers to the variational approximation $q_\phi(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})$, the generative network (decoder) parameterized by $\theta$ refers to the likelihood $p_\theta(\mathbf{o}_{t+1} \mid \mathbf{z}, \hat{\mathbf{o}}_{t+1}, \mathbf{m})$, and the standard Gaussian distribution $\mathcal{N}(0,1)$ refers to the prior $p_\theta(\mathbf{z})$. The marginal likelihood $p_\theta(\mathbf{o}_{t+1} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})$ can be rewritten in a log form:

$$\log p_\theta(\mathbf{o}_{t+1} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m}) = \mathscr{L}(\theta, \phi; \mathbf{o}_{t+1}) + KL\big(q_\phi(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m}) \,\|\, p_\theta(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1}, \mathbf{m})\big), \tag{6.8}$$

where $\mathscr{L}(\theta, \phi; \mathbf{o}_{t+1})$ is the evidence lower bound (ELBO) on the log marginal likelihood, and $KL(\cdot)$ is the Kullback–Leibler (KL) divergence function.

To optimize this log marginal likelihood (6.8), we can simply maximize the ELBO $\mathscr{L}(\theta,\phi;\mathbf{o}_{t+1})$ [137]

$$\mathscr{L}(\theta,\phi;\mathbf{o}_{t+1}) = \mathbb{E}_{q_\phi(\mathbf{z}|\hat{\mathbf{o}}_{t+1},\mathbf{m})}\left[\log p_\theta(\mathbf{o}_{t+1} \mid \mathbf{z},\hat{\mathbf{o}}_{t+1},\mathbf{m})\right] - KL\big(q_\phi(\mathbf{z} \mid \hat{\mathbf{o}}_{t+1},\mathbf{m}) \,\|\, p_\theta(\mathbf{z})\big). \tag{6.9}$$

The first term on the right-hand side (RHS) is the expected generative error, describing how well the future environment states can be generated from the latent variable $\mathbf{z}$. The second RHS term is the KL divergence, describing how close the variational approximation is to the prior.

Finally, we can use mini-batching, Monte-Carlo estimation, and reparameterization tricks to calculate the stochastic gradients of the ELBO (6.9) [137], and obtain the optimized model parameters $\phi$ and $\theta$. Using the learned model parameters $\phi$ and $\theta$, our VAE predictor network is able to integrate the predicted features $\{\hat{\mathbf{o}}_{t+1},\mathbf{m}\}$ of dynamic objects and static objects, and output a probabilistic estimate of the future environment state with uncertainty awareness.

## 6.2 Software Optimization

We demonstrated in our previous work [9] (results also in Sec. 6.3) that SCOPE and SCOPE++ are able to accurately predict maps. However, the use of multiple neural networks is resource-intensive. For example, the VAE used to generate samples to provide uncertainty estimates is so memory-intensive that we can only generate 8 samples using an NVIDIA Jetson AVG Xavier embedded device [9].[3] In order to make our proposed

[3]Generating 8 samples is a trade-off in running speed and memory consumption when running with other navigation-related packages such as `amcl`, `move_base`, and other sensor driver ROS packages.

Table 6.1. Resource utilization percentage of SCOPE++ predictor on the embedded device.

| Metrics | Ego-Motion Compensation Module | Static Object Module | Dynamic Object Module | VAE Module |
|---|---|---|---|---|
| Runtime | **9.04%** | 53.41% | 20.93% | 16.62% |
| Memory Usage | **0.32%** | 1.28% | 48.38% | 50.02% |

SCOPE/SCOPE++ predictors work efficiently in different real-world robotics applications, we decide to optimize them on the software side. Compared with hardware optimization, the benefit of software optimization is that there are no requirements for specific computing devices (such as FPGAs and ASICs), and currently most robots are not equipped with these computing devices.

### *6.2.1 Resource Utilization*

Table 6.1 details the running performance percentage of each module in the SCOPE++ predictor (generating 8 samples) in terms of runtime and memory usage on our embedded computer. We can easily see that the runtime is mainly limited by the static object module and dynamic object module (with the ConvLSTM unit), while the memory usage is mainly limited by the dynamic object module (with ConvLSTM unit) and VAE module. This analysis explains why our proposed SCOPE predictor without the static object module is much faster than our SCOPE++ predictor.

Since the key ConvLSTM unit in the dynamic object module is already an accelerated RNN unit (compared to the Vanilla RNN unit), it is difficult to further optimize this module. Therefore, we choose to perform software optimization on the static object module and VAE module to increase running speed and reduce memory consumption while

maintaining accurate prediction performance and the stochastic nature of our proposed SCOPE/SCOPE++ predictors.

According to Table 6.1 and our network architecture design, a straightforward method is to remove the static object module (*i.e.,* use SCOPE instead of SCOPE++), especially since the two variants achieve comparable prediction accuracy. Next, we will look at replacing the VAE network by utilizing knowledge distillation techniques to compress our SCOPE network to maintain its prediction performance, and then quantify the prediction uncertainty of the SCOPE to provide uncertainty estimates to maintain its stochastic nature.

### *6.2.2 Knowledge Distillation*

Compared with other deep neural network compression techniques (*e.g.,* pruning, quantization), knowledge distillation techniques can provide better generalization capabilities and allow the selection of different network architectures [141]. To design a simple and efficient student network for knowledge distillation, we remove the static object module (following our fast SCOPE network architecture) and replace the complete VAE network with a single convolutional layer, which can be seen from the SO-SCOPE part of Fig. 6.2. Therefore, our SO-SCOPE network is very simple and efficient, consisting of only one ConvLSTM layer and one convolutional layer.

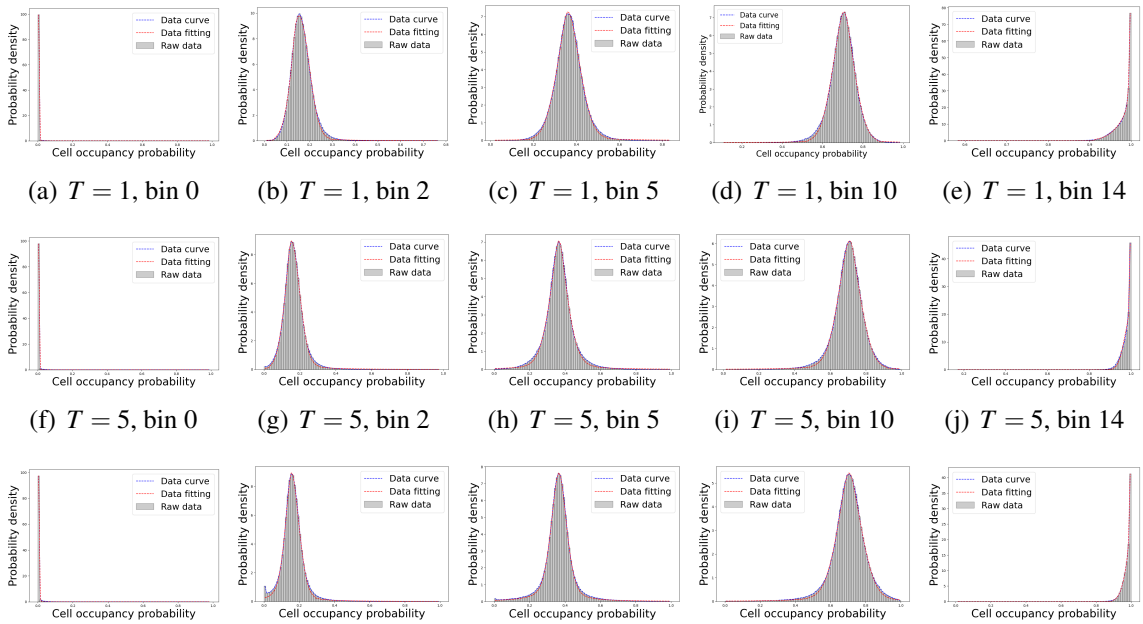
We use the SCOPE network instead of the SCOPE++ network as our teacher network for knowledge distillation, shown in Fig. 6.2. There are two reasons why we choose the fast and less accurate SCOPE network rather than the complex and more accurate SCOPE++ network as the teacher network. First, as Cho et al. [48] suggested, larger or more accurate models generally do not make better teachers in the knowledge distillation, especially when

the student's ability is too low to successfully imitate the teacher. Second, Beyer et al. [142] suggested that keeping "consistent" teaching can improve knowledge distillation. Our SO-SCOPE student network follows the SCOPE network design, which is almost the same as the SCOPE network except that a single convolutional layer is used instead of the VAE network (*i.e.,* the input view of the convolutional layer is consistent with the VAE network). Therefore, this structural similarity and consistency improves knowledge transfer from the teacher network to the student network. This will accelerate the running speed and also reduce the memory consumption.

To train our SO-SCOPE student network, we directly use the output of our trained SCOPE teacher model as "soft" labels instead of using ground truth labels, which follow the key idea of knowledge distillation. Note that since our SCOPE predictor is stochastic and generates different samples even with the same input, this stochastic sample output expands the original training dataset and further improves the generalization ability of the SO-SCOPE predictor. We use the binary cross entropy (BCE) loss function to train our SO-SCOPE student network and get our final SO-SCOPE predictor to provide deterministic OGM predictions. After completing the knowledge distillation training, we directly use the output of the SO-SCOPE predictor to provide OGM prediction information. However, this distilled network is now deterministic, *i.e.,* each input will generate a single output. Our next step will address this limitation.

### *6.2.3 Uncertainty Quantification*

To modify SO-SCOPE, we will characterize the statistics of the VAE from our trained SCOPE++ predictor and then combine this data with the output of the distillation network




(a) $T = 1$, bin 0 (b) $T = 1$, bin 2 (c) $T = 1$, bin 5 (d) $T = 1$, bin 10 (e) $T = 1$, bin 14

(f) $T = 5$, bin 0 (g) $T = 5$, bin 2 (h) $T = 5$, bin 5 (i) $T = 5$, bin 10 (j) $T = 5$, bin 14

(k) $T = 10$, bin 0 (l) $T = 10$, bin 2 (m) $T = 10$, bin 5 (n) $T = 10$, bin 10 (o) $T = 10$, bin 14

Figure 6.3. Uncertainty quantification visualization of predicted OGM cell bins for different prediction timesteps. Grey histograms are the raw predicted OGM cell data of our SCOPE++ predictor. Blue curves are the connecting curves of these raw predicted OGM cell data. Red curves are the fitting curves from our proposed mixture model of truncated normal and skew-Cauchy distributions.

(which should generate an OGM that is similar to an output of the VAE) to draw samples with identical statistics to our original network but in a much more memory- and time-efficient manner. Note that using SOGMP++ here is a reasonable choice because our SCOPE++ predictor provides a more accurate and comprehensive uncertainty estimate than SCOPE. To characterize the VAE, we collected 8000 different input sequences to the ConvLSTM from the OGM-Turtlebot2 test dataset (see Sec. 6.3.1) and generated 32 VAE samples for each input, giving us a total of $256,000$ output samples for each prediction horizon $T = 1, \ldots, \tau$.

Our analysis relies on two factors. First, in an OGM, the probability of occupancy for each cell is independent of all other cells [140]. In other words, for two cells $c$ and $c'$, then

$p(c \text{ is occupied and } c' \text{ is occupied}) = p(c \text{ is occupied})p(c' \text{ is occupied})$. Second, based on our use of a VAE, we make the assumption that the distribution of occupancy probabilities in the VAE output depends on the occupancy probability of the input. This is reasonable since a VAE is trained to have similar outputs be near to one another in the learned latent space. Let $\hat{c}_{t+T} = p(c \text{ is occupied at time } t+T)$ denote the probability of occupancy of cell $c$ predicted by the ConvLSTM layer $T$ steps into the future (*i.e.,* we use the average of the occupancy probabilities predicted by the VAE for the same cell as its estimate) and $\tilde{c}_{t+T}$ be the probability of occupancy predicted for that same cell by the VAE. Then we assume $p(\tilde{c}_{t+T} \mid \hat{c}_{t+T}) \neq p(\tilde{c}_{t+T})$.

Based on these factors, we will examine trends in the VAE probabilities $\tilde{c}_{t+T}$ (for $T = 1, \ldots, \tau$) conditioned on the value of $\hat{c}_{t+T}$. We use 15 bins for $\hat{c}_{t+T}$, evenly spaced from 0 to 1 (since the probability range of OGM cell is in $[0,1]$). With these binned OGM cell samples, we visualize the distribution of $\tilde{c}_{t+T}$ each $\hat{c}_{t+T}$ bin, as shown in Fig. 6.3. We can see a few useful trends in the data. First, cells that are predicted to be empty by the ConvLSTM (*i.e.,* in bin 0) remain empty for nearly all VAE samples. This trend holds during the full prediction window, with only a slight increase in the probability of occupancy at $T$ getting larger. Second, we see that the peak of the distribution for $\tilde{c}_{t+T}$ remains very close to the value of $\hat{c}_{t+1}$. This implies that the VAE generates a consistent distribution around the ConvLSTM output. Third, we see heavier tails in the distributions as $T$ increases. This makes sense as a longer horizon into the future leads to further uncertainty and further deviation from the current time. Fourth, we see the distributions skew towards 0 as the prediction horizon $T$ increases. This makes sense as cells are more likely to become empty as people move about in the scene. Fifth, for situations where $\hat{c}_{t+1}$ is low, we see a

peak in the distribution of $\tilde{c}_{t+T}$ at 0. This is caused by uncertain objects moving out of a cell, leaving it empty.

Our goal is to fit a model to the resulting histograms. Our model must be able to capture the heavy tails, skew, and the peak near $\tilde{c} = 0$. Based on this, we use a mixture model of a truncated normal and a skew-Cauchy distribution to describe the distribution of $\tilde{c}_{t+T}$ values, which can be expressed mathematically as:

$$f_{\xi}(\tilde{c}) = w \frac{\sqrt{\frac{2}{\pi\sigma_{\text{tn}}^2}} \exp\left(-\frac{1}{2}\left(\frac{\tilde{c}-\mu_{\text{tn}}}{\sigma_{\text{tn}}}\right)^2\right)}{\text{erf}\left(\frac{b-\mu_{\text{tn}}}{\sqrt{2}\sigma_{\text{tn}}}\right) - \text{erf}\left(\frac{a-\mu_{\text{tn}}}{\sqrt{2}\sigma_{\text{tn}}}\right)} + (1-w) \frac{1}{\sigma_{sc}\pi\left[1 + \left(\frac{|\tilde{c}-\mu_{sc}|}{\sigma_{sc}(1+\lambda\,\text{sign}(\tilde{c}-\mu_{sc}))}\right)^2\right]}, \quad (6.10)$$

where the first term on RHS is the truncated normal distribution (with mean $\mu_{\text{tn}}$, standard deviation $\sigma_{\text{tn}}$, lower bound $a$, and upper bound $b$), the second RHS term is the skew-Cauchy distribution (with location parameter $\mu_{\text{sc}}$, scale parameter $\sigma_{\text{sc}}$, and skewness $\lambda$), and $w \in [0,1]$ is the mixture weight.

Let $\xi = [w, a, b, \mu_{\text{tn}}, \sigma_{\text{tn}}, \lambda, \mu_{\text{sc}}, \sigma_{\text{sc}}]$ be the vector of mixture model parameters. We use a least square optimization algorithm [143] to fit $\xi$ to our predicted OGM cell bin data. Figure 6.3 shows the results of this fit. We see that the model achieves strong performance across a range of bins and timesteps, indicating that our methods work well.

The end result of our analysis above is a set of 150 $\xi$ vectors, one for each of the 15 $\hat{c}$ bins and each of the 10-time steps $T$. For each set of mixture model parameters, we can generate different types of distribution statistics (*e.g.,* mean, median, mode, entropy, and random variate) and store these values in a lookup table. We call this lookup table the prediction uncertainty statistics lookup table.

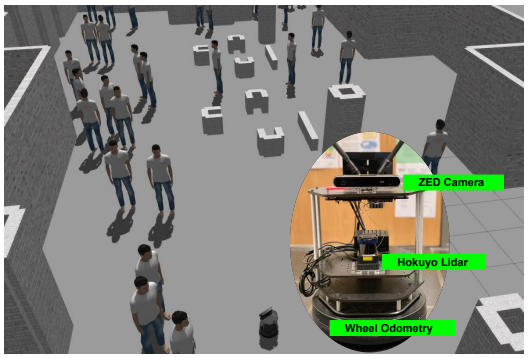
Now that we have accurate models for the statistics of how each cell in the map will change over time, we can integrate that information into the SO-SCOPE model. To do this, we use the output of the distillation layer to get $\hat{c}_{t+T}$ for each cell in the OGM $\mathbf{o}_{t+T}$, as shown in the SO-SCOPE part of Fig. 6.2. We can then use the prediction uncertainty statistics lookup table to get statistics of that cell, or we can use the cumulative distribution function of (6.10) to draw samples $\tilde{c}_{t+T}$. This allows our SO-SCOPE model to account for uncertainty in the predicted future output using knowledge distilled from the SCOPE and SCOPE++ models to achieve comparable accuracy with significantly smaller computational effort and memory consumption.

## 6.3 OGM Prediction Experiments and Results

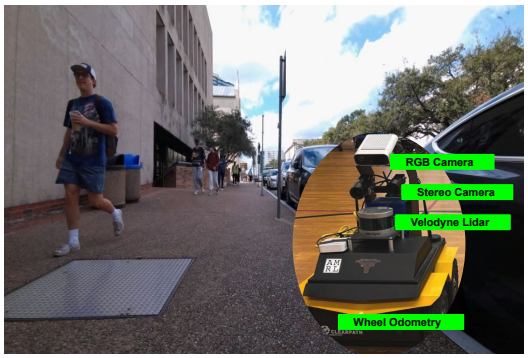
To demonstrate the prediction performance of our proposed approaches, we first test our algorithms on a simulated dataset and two public real-world sub-datasets from the socially compliant navigation dataset (SCAND) [144]. We comprehensively evaluate the inference speed and memory usage of our proposed predictors on a resource-constrained embedded computing device to show that our software-optimized predictor can achieve surprising performance improvements. We also characterize the uncertainty of our proposed predictors across different sample sizes and numbers of objects to demonstrate the diversity and consistency of their uncertainty estimates.

### *6.3.1 Datasets*

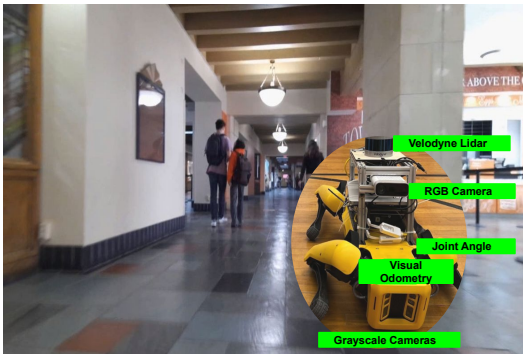
We used three publically available OGM datasets to evaluate our proposed prediction algorithms and baselines, one self-collected from the Gazebo simulator (*i.e.,* OGM-




(a) OGM-Turtlebot2 dataset




(b) OGM-Jackal dataset




(c) OGM-Spot dataset

Figure 6.4. Illustration of the OGM dataset collection from three different robots.

Turtlebot2) [145] and two extracted from the real-world dataset SCAND [144] (*i.e.,* OGM-Jackal and OGM-Spot).

**OGM-Turtlebot2 Dataset.** To train our predictor networks and baselines, we collected the OGM-Turtlebot2 dataset using the 3D robot-pedestrian interaction Gazebo simulator from our previous work [6, 8]. Specifically, a simulated Turtlebot2 equipped with a 2D Hokuyo UTM-30LX lidar navigates around an indoor environment with 34 moving pedestrians using random start points and goal points, as shown in Fig. 6.4(a). The Turtlebot2 uses the dynamic window approach (DWA) planner [5] and has a maximum speed of 0.8 m/s. The pedestrians are driven by the social force model [107] with an average speed of 1.5 m/s. We collected the robot states $\{\mathbf{x}, \mathbf{u}\}$ and raw lidar measurements $\mathbf{y}$ at a sam-

pling rate of 10 Hz. We collected a total of 94,891 $(\mathbf{x}, \mathbf{u}, \mathbf{y})$ tuples, dividing this into three separate subsets for training (67,000 tuples), validation during training (10,891 tuples), and final testing (17,000 tuples).

**OGM-Jackal and OGM-Spot Datasets.** Fig. 6.4(b) shows the real-world outdoor environment at UT Austin and the Jackal robot used to collect the raw SCAND dataset to construct the OGM-Jackal dataset. Fig. 6.4(c) shows the real-world indoor environment at UT Austin and the Spot robot used to collect the raw SCAND dataset to construct the OGM-Spot dataset. Note that the SCAND dataset was collected by humans manually operating the Jackal robot and the Spot robot around the indoor/outdoor environments at UT Austin. More details can be found in [144].

### *6.3.2 OGM Prediction*

We compare our proposed SCOPE, SCOPE++, and SO-SCOPE algorithms with four deep learning-based baselines: ConvLSTM [94], DeepTracking [88], PhyDNet [96], and five ablation baselines: SCOPE-NEMC (SCOPE with No Ego-Motion Compensation module), SO-SCOPE-MEAN (use the mean value from the prediction uncertainty statistics lookup table as the OGM prediction), SO-SCOPE-MEDIAN (use the median from the lookup table), SO-SCOPE-MODE (use the mode from the lookup table), and SO-SCOPE-SAMPLE (draw a random sample from distribution). Note that all these networks were implemented using PyTorch framework [126] and trained using *only* our self-collected OGM-Turtlebot2 dataset.

**Evaluation Metrics.** We use three error metrics: 1) weighted mean square error (WMSE) [146] to evaluate absolute error between the ground truth and predicted maps, 2)

structural similarity index measure (SSIM) [147] to measure the similarity between the two maps, and 3) optimal subpattern assignment metric (OSPA) [148] to measure the difference in the number and locations of distinct objects within the maps. Specifically, if we define the predicted OGM as $\bar{\mathbf{o}}$ and the ground truth OGM as $\mathbf{o}$, then these three evaluation metrics can be mathematically expressed as:

- **Weighted mean square error (WMSE) [146]**:

$$WMSE = \frac{\sum_{i=1}^{N} w_i \left(\bar{\mathbf{o}}_i - \mathbf{o}_i\right)^2}{\sum_{i=1}^{N} w_i}, \tag{6.11}$$

  where $N$ is the number of cells in the OGM, and $w_i$ is the weight for the cell $i$ in the OGM, calculated by the median frequency balancing method [149]. This metric is used to evaluate the weighted absolute errors (balancing the imbalance in the percentage of occupied and free cells) between the predicted OGM and its corresponding ground truth OGM, describing the predicted quality of a single OGM cell.

- **Structural similarity index measure (SSIM)** [147]:

$$SSIM = \frac{\left(2\mu_{\bar{\mathbf{o}}}\mu_{\mathbf{o}} + C_1\right)\left(2\delta_{\bar{\mathbf{o}}\mathbf{o}} + C_2\right)}{\left(\mu_{\bar{\mathbf{o}}}^2 + \mu_{\mathbf{o}}^2 + C_1\right)\left(\delta_{\bar{\mathbf{o}}}^2 + \delta_{\mathbf{o}}^2 + C_2\right)}, \tag{6.12}$$

  where $\mu_{(\cdot)}$ and $\delta_{(\cdot)}$ denote the mean and variance/covariance, respectively, and $C_{(\cdot)}$ denotes constant parameters to avoid instability. We use $C_1 = 1\text{e}{-4}$ and $C_2 = 9\text{e}{-4}$. This metric is used to evaluate the structural similarity between the predicted OGM and its corresponding ground truth OGM, describing the predicted quality of the scene geometry.

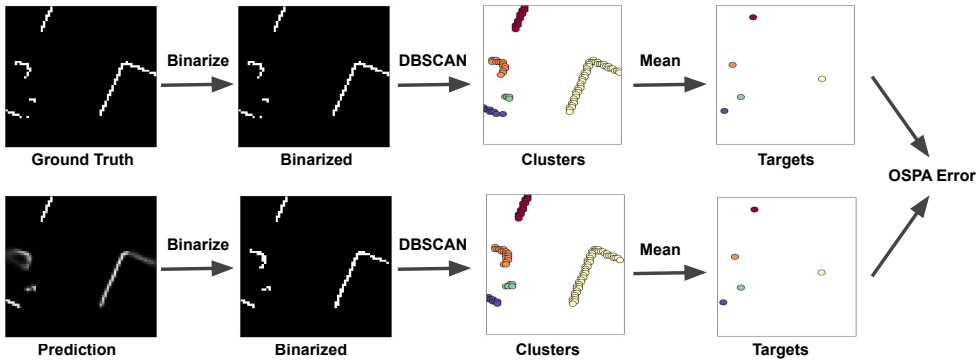



Figure 6.5. Evaluation pipeline for calculating OSPA error on predicted OGMs.

- **Optimal subpattern assignment metric (OSPA)** [148]:

$$OSPA = \left( \frac{1}{n} \min_{\pi \in \Pi_n} \sum_{1}^{m} d_c(\bar{\mathbf{o}}_i, \mathbf{o}_{\pi(i)})^p + c^p(n-m) \right)^{\frac{1}{p}}, \tag{6.13}$$

where $c$ is the cutoff distance, $p$ is the norm associated to distance, $d_c(\bar{\mathbf{o}}, \mathbf{o}) = \min(c, \|\bar{\mathbf{o}} - \mathbf{o}\|)$, and $\Pi_n$ is the set of permutations of $\{1, 2, ..., n\}$. We use $c = 10$ (*i.e.,* 1 m) and $p = 1$. This metric is used to evaluate the target tracking accuracy between the predicted OGM and its corresponding ground truth OGM, describing the predicted quality of multi-target localization and assignment.

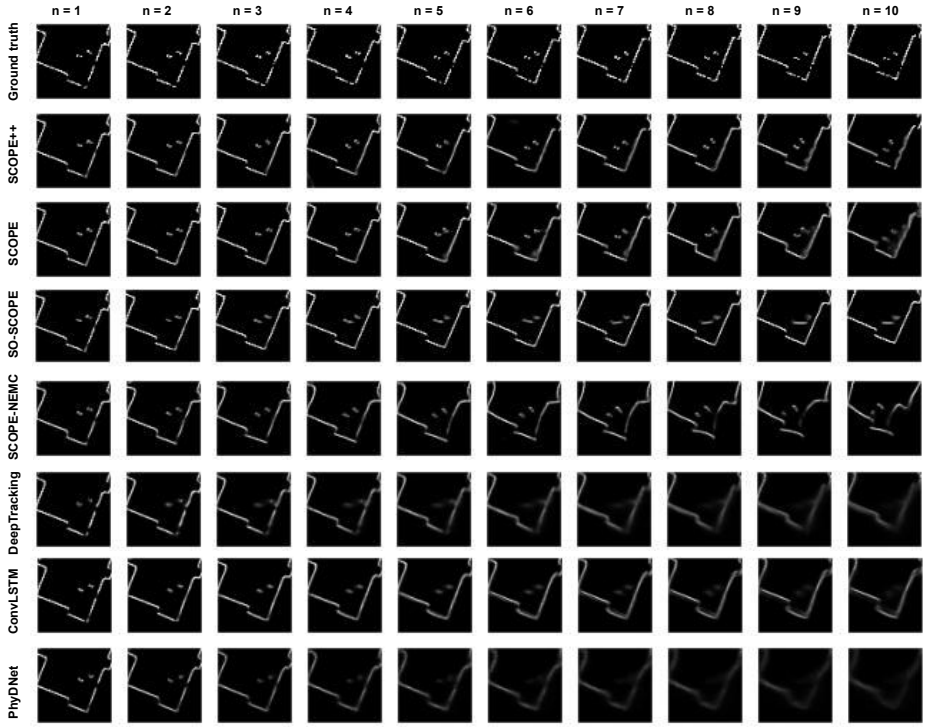
Note that other OGM prediction works [89–93, 97–99, 102] only use the computer vision metrics (*e.g.,* MSE, F1 Score, SSIM, and confusion matrices) to evaluate the quality of predicted OGMs. We are the first to evaluate the predicted OGMs from the perspective of multi-target tracking (*i.e.,* OSPA distance), which we believe gives a more physically meaningful evaluation than image quality as it accounts for the number of and locations of objects in the scene.

Figure 6.5 outlines our process to calculate the OSPA. First, we binarize the predicted and ground truth OGMs with an occupancy threshold $p_{\text{free}} = 0.3$, which is set by referenc-

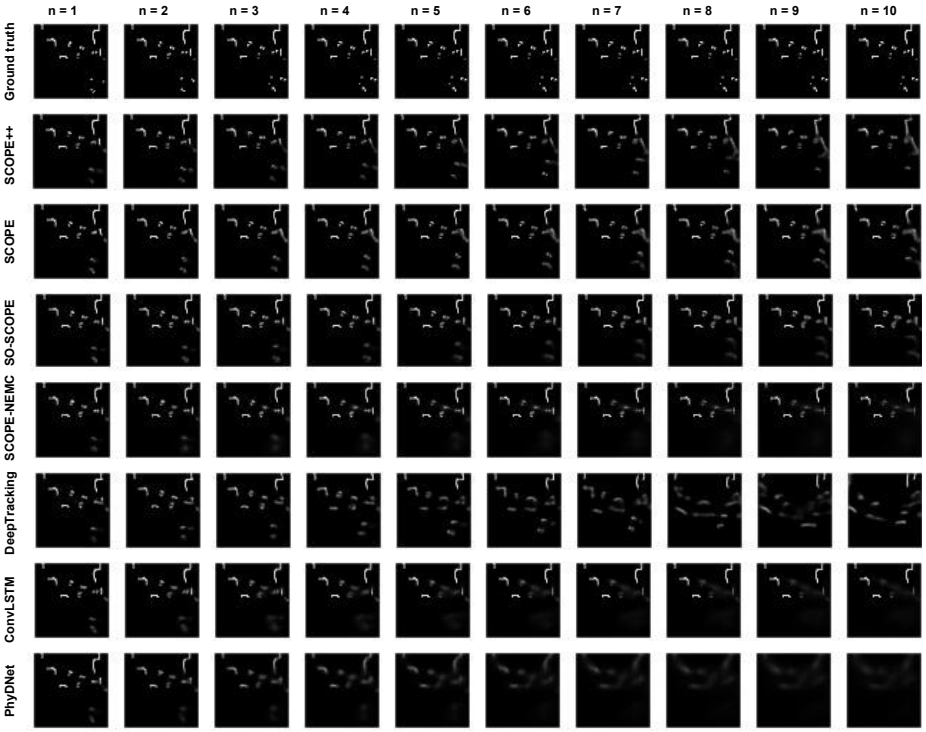
ing the `occ_thresh` default parameter of 0.25 from the `gmapping` ROS package. Second, we use the density-based spatial clustering of applications with noise (DBSCAN) [150] algorithm to cluster the obstacle points in the OGMs. This yields estimated and ground truth sets of object locations in the scene. We then apply the OSPA error metric [148] between these object sets (with cutoff distance 10 cells, or 1 m).

**Qualitative Results.** Figure 6.6 and the accompanying Multimedia illustrate the future OGM predictions generated by our proposed predictors and the baselines. We observe three interesting phenomena. First, the image-based baselines, especially the PhyDNet, generate blurry future predictions after 5-time steps, with only blurred shapes of static objects (*i.e.,* walls) and missing dynamic objects (*i.e.,* pedestrians). We believe that this is because these three baselines are deterministic models that use less expressive network architectures, only treat time series OGMs as images/video, and cannot capture and utilize the kinematics and dynamics of the robot itself, dynamic objects, and static objects. Second, the SCOPE++ with a local environment map has a sharper and more accurate surrounding scene geometry (*i.e.,* right walls) than the SCOPE without a local environment map. This difference indicates that the local environment map for static objects is beneficial and plays a key role in predicting surrounding scene geometry. Third, our proposed software-optimized SO-SCOPE predictor is able to achieve clear and sharp OGM predictions similar to its "teacher" SCOPE predictor, which demonstrates the effectiveness of applying knowledge distillation techniques to optimize our SCOPE++ predictor.

**Quantitative Results.** While achieving sharper predictions is encouraging, we also care about the quantitative measures of our proposed approaches. We test the OGM predictors on the OGM-Turtlebot2 test dataset, OGM-Jackal dataset, and OGM-Spot dataset by




(a) Structural objects



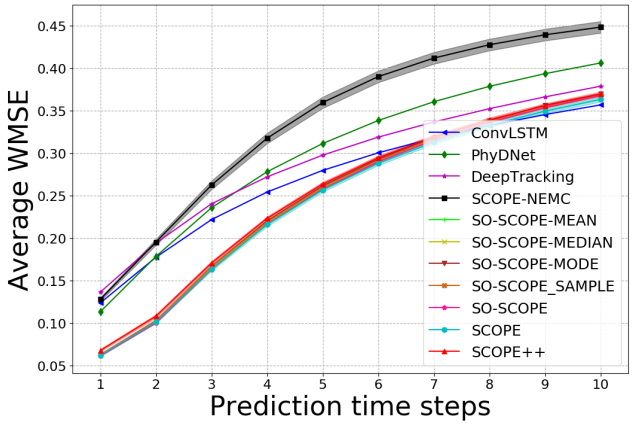

(b) Cluttered objects

Figure 6.6. Two Prediction showcases of seven OGM predictors tested on the OGM-Turtlebot2 dataset over the prediction horizon. The black area is the free space and the white area is the occupied space.

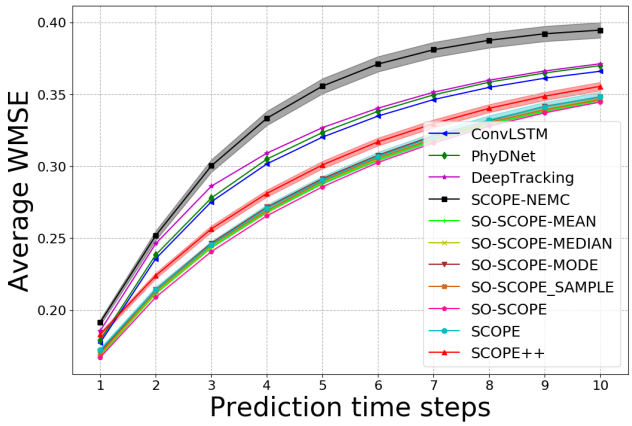
evaluating absolute error, structure similarity, and tracking accuracy. Recall that we only use the OGM-Turtlebot2 during training, so the other two datasets are entirely new and were generated by robots with different speeds and kinematics (*i.e.,* Spot ambulates).

**1) Absolute Error.** To evaluate the absolute error of predicted OGMs, we calculate the average WMSE of predicted OGMs for the next 10 prediction time steps, outlined in Fig. 6.7. It can be seen that our proposed SCOPE family with ego-motion compensation achieves significantly better average WMSE, SSIM, and OSPA than the SCOPE-NEMC ablation baseline without ego-motion compensation in all test datasets, which illustrates the effectiveness of ego-motion compensation. Furthermore, the average WMSEs of our proposed SCOPE family of methods at different prediction time steps are not significantly different from one another, but are all lower than the three image-based baselines (*i.e.,* ConvLSTM [94], DeepTracking [88], and PhyDNet [96]). This trend holds across all three test datasets collected by different robots. This shows that the proposed SCOPE family of methods utilizing kinematic and dynamic information are able to predict more accurate OGMs than the state-of-the-art image-based approaches at different prediction time steps.

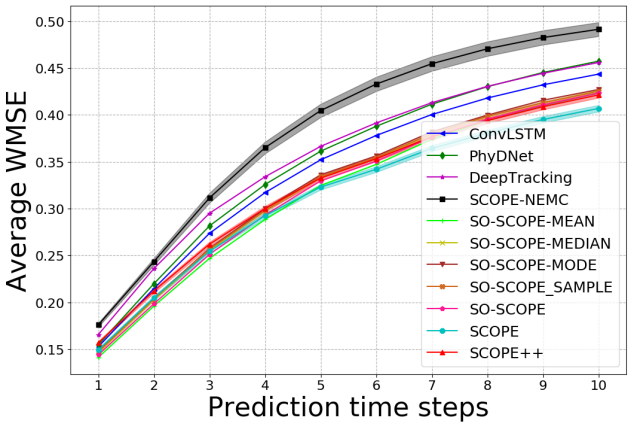
**2) Structure Similarity.** To evaluate the structure similarity of predicted OGMs, we calculate the average SSIM of predicted OGMs for the next 10 prediction time steps, as shown in Fig. 6.8. Unlike general WSME, which cares about the absolute error of a single OGM cell, SSIM cares about the entire OGM structure and its scene geometry. Figure 6.8 shows three interesting phenomena. First, in all three different test datasets, the average SSIMs of our proposed SCOPE, SCOPE++, and SO-SCOPE predictors at different prediction time steps are significantly higher than that of the other three image-based baselines and SCOPE-NEMC, which further illustrates the effectiveness of the ego-motion




(a) OGM-Turtlebot2




(b) OGM-Jackal



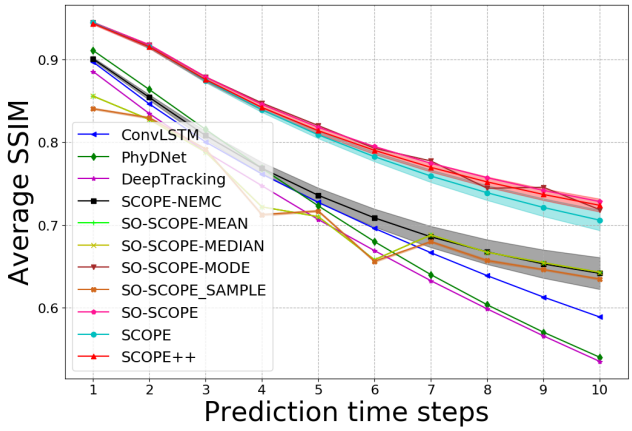

(c) OGM-Spot

Figure 6.7. Average WMSE of 10 different prediction time steps for all tested methods on different datasets. Note that the average WMSE shadows for SCOPE-NEMC, SCOPE, and SCOPE++ approaches are plotted with a 95% confidence interval over 32 samples. For all curves, lower is better.

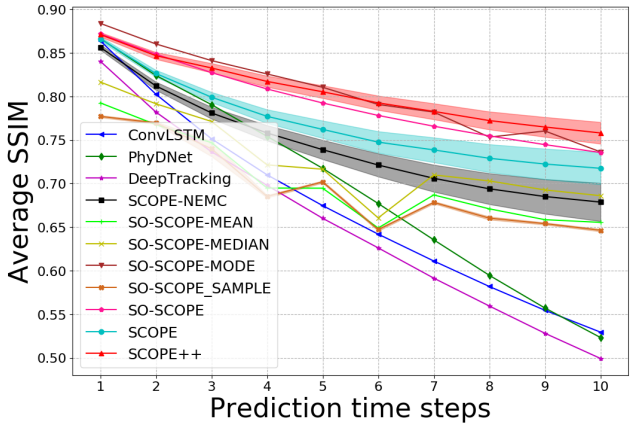
compensation. Second, the average SSIMs of our software-optimized SO-SCOPE predictor and its statistical ablation baseline SO-SCOPE-MODE (*i.e.,* using peak values as cell prediction values) are not significantly different (similar to that of the SCOPE++ predictor), but much higher than the other three statistical ablation baselines (*i.e.,* SO-SCOPE-MEAN, SO-SCOPE-MEDIAN, SO-SCOPE-SAMPLE). It shows that the direct output of SO-SCOPE "student" network is able to represent the prediction information and the mode from our prediction uncertainty statistics lookup table is also a good statistic to represent the prediction information (compared to other statistics like mean, median, and random variate), which further supports the choice of using knowledge distillation techniques and uncertainty quantification to software optimize our original SCOPE/SCOPE++ predictors. Third, it is interesting that the average SSIM of our SCOPE++ with a local environment map is significantly higher than that of SCOPE without a local environment map in long-term predictions over multiple time steps. This indicates that the local environment map that takes into account static objects helps to predict the long-term future of the environment. This is reasonable because the local environment map describes the basic scene geometry and shape.

**3) Tracking Accuracy.** To further evaluate the tracking accuracy of predicted OGMs, we calculate the average OSPA of predicted OGMs for the next 10 prediction time steps, as shown in Fig. 6.9. While the average WMSE and SSIM are the evaluation metrics from computer vision, the average OSPA is from the multi-target tracking and is more suitable for evaluating the physical quality of OGMs.

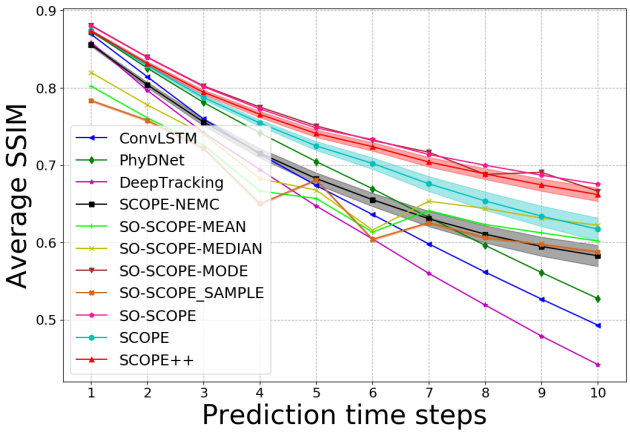
Figure 6.9 reveals that the average OSPA errors of our proposed SCOPE, SCOPE++, and SO-SCOPE predictors are significantly lower than the average OSPA errors of the




(a) OGM-Turtlebot2




(b) OGM-Jackal

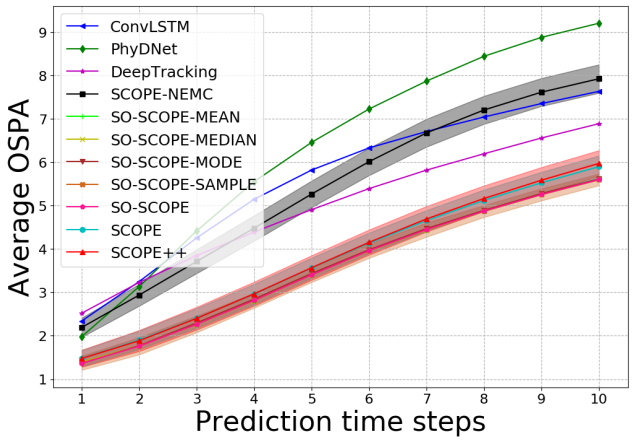



(c) OGM-Spot

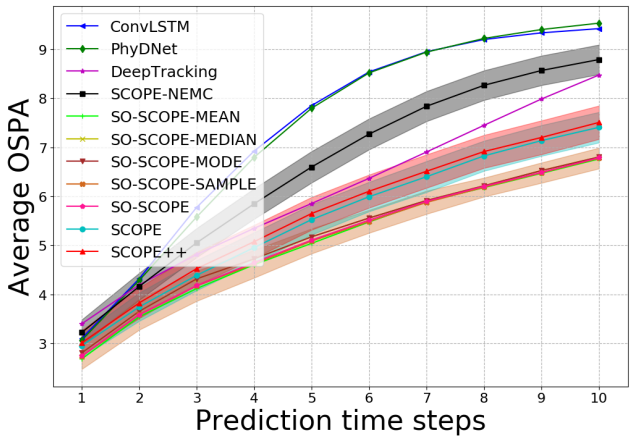
Figure 6.8. Average SSIM of 10 different prediction time steps for all tested methods on different datasets. Note that the average SSIM shadows for SCOPE-NEMC, SCOPE, and SCOPE++ approaches are plotted with 95% confidence interval over 32 samples. For all curves, higher is better.

other three image-based baselines and SCOPE-NEMC, especially in longer prediction time steps. This exciting result further demonstrates the preferential performance of our proposed motion-based methods (*i.e.,* SCOPE, SCOPE++, and SO-SCOPE) in tracking or predicting environmental states (*i.e.,* localization and cardinality) over other image-based methods (*i.e.,* ConvLSTM [94], DeepTracking [88], and PhyDNet [96]). In addition, the average OSPA errors of our software-optimized SO-SCOPE predictor and its statistical ablation baselines are not significantly different from one another (similar to the WMSE metric), but are lower than the original SCOPE and SCOPE++ preditor (especially in the real-world OGM-Jackal and OGM-Spot dataset), which is reasonable and consistent with the argument in many knowledge distillation works [45, 141] that the "student" network can generalize better than the "teacher" network. However, the average OSPA errors of our proposed SCOPE and SCOPE++ predictors are almost the same, where the local environment map for static objects does not help SCOPE++ to reduce the OSPA error. We believe that this is because the local environment map only provides useful information for static objects while the OSPA metric is biased towards dynamic objects (*e.g.,* pedestrians) since there tend to be more of them than static objects (*e.g.,* walls).

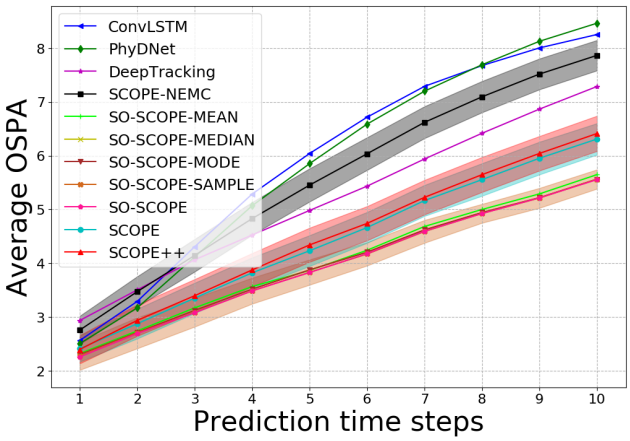
**4) Summary.** Taken together, these meaningful results on the absolute error of a single occupancy grid cell, the structure similarity of overall scene geometry, and the tracking accuracy of multi-target localization and cardinality demonstrate that our proposed SCOPE, SCOPE++, and SO-SCOPE predictors with uncertainty estimates can predict sharper and more accurate long-term future states of the environment than image-based baselines.




(a) OGM-Turtlebot2




(b) OGM-Jackal




(c) OGM-Spot

Figure 6.9. Average OSPA of 10 different prediction time steps for all tested methods on different datasets. Note that the average OSPA shadows for SCOPE-NEMC, SCOPE, and SCOPE++ approaches are plotted with a 95% confidence interval over 32 samples. For all curves, lower is better.

Table 6.2. Inference speed and model size of different OGM predictors.

| **Models** | ConvLSTM [94] | PhyDNet [96] | DeepTracking [88] | SCOPE-NEMC | SCOPE | SCOPE++ | SO-SCOPE |
|---|---|---|---|---|---|---|---|
| **Inference Speed (FPS)** | 2.95 | 4.66 | 5.32 | 24.83 | 23.29 | 10.68 | **34.75** |
| **Model Size (MB)** | 12.44 | 37.17 | **0.95** | 8.84 | 8.84 | 8.85 | 1.80 |

### *6.3.3 Computational Resource Utilization*

While it is encouraging that our proposed SCOPE family of OGM predictors exhibits good prediction performance, we also want to evaluate the inference speed and model size of all tested OGM predictors. This is because mobile robots are resource-limited, and smaller model sizes and faster inference speeds mean robots have a faster reaction time to face and handle dangerous situations in complex dynamic scenarios.

**Baseline Comparison.** Table 6.2 summarizes the inference speed and model size of six predictors tested on an NVIDIA Jetson TX2 embedded computer equipped with a 256-core NVIDIA Pascal @ 1300MHz GPU and loaded with 8GB of memory. We can see that: 1) DeepTracking [88] has the smallest model size, our software-optimized SO-SCOPE model is the next smallest, our SCOPE and SCOPE++ models are about 4.9 times larger than SO-SCOPE, ConvLSTM [94] is about 1.4 times larger than these, and PhyDNet [96] is about 3 times larger than ConvLSTM. 2) Our proposed SCOPE family achieves faster inference than all other state-of-the-art baselines, especially our SO-SCOPE which can run at up to 35 FPS. These show that our proposed SCOPE family (*i.e.,* SCOPE++, SCOPE, and SO-SCOPE) can have faster inference speed and reasonable model size, which is more hardware-friendly than other state-of-the-art OGM predictors.

**SCOPE Family Detailed Comparison.** Then, we comprehensively compare the inference speed and memory usage of our software-optimized SO-SCOPE predictor with

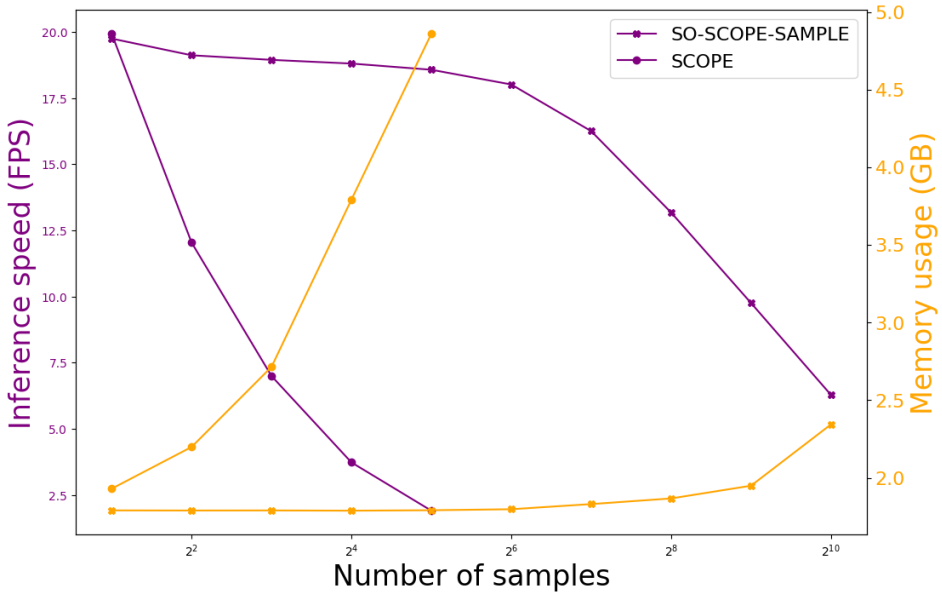



Figure 6.10. Inference speed and memory usage of SCOPE and SO-SCOPE predictors with different numbers of samples.

our SCOPE predictor (*i.e.,* a fast version of our SCOPE++) on a resource-constrained embedded computing device to show how our proposed software optimization approaches improve the embedded operation performance of SCOPE and SCOPE++. For a fair comparison with our SCOPE predictor, we set the SO-SCOPE predictor into a mode that generates samples where the prediction uncertainty statistics lookup table will provide the random variables. We compared their inference speed and memory usage when generating different numbers of samples, increasing as a power of 2 in the range $[2, 1024]$. Figure 6.10 shows their detailed comparison results.

As can be seen from Fig. 6.10, the inference speed of the SCOPE predictor drops significantly as the number of samples increases, with the lowest inference speed being about 2 FPS. In contrast, the inference speed of the SO-SCOPE hardly slows down as the number of samples increases until 128 samples are generated, and even with 1,024 samples it can still achieve around 6 FPS (*i.e.,* the speed bottleneck is generating a large number

of samples rather than network inference). This demonstrates the significant advantage of using knowledge distillation techniques to compress our SCOPE++ network, since smaller networks lead to faster inference speeds, and using prediction uncertainty statistics lookup table to provide uncertainty estimates for our SO-SCOPE predictor, which speeds up the time-consuming sample generation process. Note that we could parallelize OGM sampling using our SO-SCOPE approach since it simply requires querying the lookup table, while SCOPE requires running the VAE model.

Figure 6.10, also shows that the memory usage of the SCOPE predictor increases significantly with the number of samples and can only generate up to 32 samples on the NVIDIA Jetson TX2 embedded computer due to the 8 GB memory limit. In contrast, the memory usage of the SO-SCOPE predictor barely increases with the number of samples until 1,024 samples are generated (*i.e.,* the memory bottleneck is storing a large number of samples). This demonstrates that using a prediction uncertainty statistics lookup table in our proposed SO-SCOPE predictor instead of using a VAE network in the SCOPE predictor to generate prediction samples to provide uncertainty estimates has a significant advantage in terms of memory reduction.

### *6.3.4 Uncertainty Characterization*

To characterize the uncertainty information of our SCOPE, SCOPE++, and SO-SCOPE predictors, we use the Shannon entropy [151] to measure the uncertainty. For an OGM $\mathbf{o}$, the Shannon entropy is given by

$$H(\mathbf{o}) = \sum_{c \in \mathbf{o}} H(c) = -\sum_{c \in \mathbf{o}} p \log p + (1-p)\log(1-p), \tag{6.14}$$

where the first equality comes from the assumption that cells $c$ within an OGM are independent and the second equality comes from the definition of Shannon entropy for a Bernoulli distribution with parameter $p$ (since each cell can either be occupied, with probability $p$, or free, with probability $1-p$).

In our case, we have a predicted future map, $\hat{\mathbf{o}}$ and data about the distribution of occupancy values $\tilde{c}$, which comes from the predicted occupancy value $\hat{c}$ along with the mixture model (6.10) and learned parameters $\xi$. In this case, the entropy becomes

$$H(\hat{\mathbf{o}}) = \sum_{c \in \hat{\mathbf{o}}} E_{f_\xi(\hat{c})}[H(\tilde{c})], \tag{6.15a}$$

$$= -\sum_{c \in \hat{\mathbf{o}}} \int_0^1 [\tilde{c}\log\tilde{c} + (1-\tilde{c})\log(1-\tilde{c})]\, f_\xi(\tilde{c}) d\tilde{c}, \tag{6.15b}$$

$$\approx -\sum_{c \in \hat{\mathbf{o}}} \frac{1}{M} \sum_{j=1}^{M} \left[\tilde{c}_j \log\tilde{c}_j + (1-\tilde{c}_j)\log(1-\tilde{c}_j)\right], \tag{6.15c}$$

where $M$ is the number of predicted OGM samples and $\tilde{c}_j$ is the occupancy value of the cell in the $j$th predicted OGM $\tilde{\mathbf{o}}_j$.

Note that we can directly use (6.15b) to calculate the predicted OGM entropy of SCOPE, SCOPE++, and SO-SCOPE because we have the distribution $f_\xi(c)$ for OGM cell $c$ from (6.10). We use the predicted mean map from OGM samples $\bar{\mathbf{o}} = \frac{1}{M}\sum_{j=1}^{M} \tilde{\mathbf{o}}$ instead of $\hat{\mathbf{o}}$ for SCOPE and SCOPE++, and the direct output of the SO-SCOPE network as $\hat{\mathbf{o}}$ for SO-SCOPE. On the other hand, we also use (6.15c) to estimate the predicted OGM entropy of SCOPE and SCOPE++, since we can obtain samples of predicted OGM through the VAE module. We also normalize the data by dividing the entropy by the total number of cells.

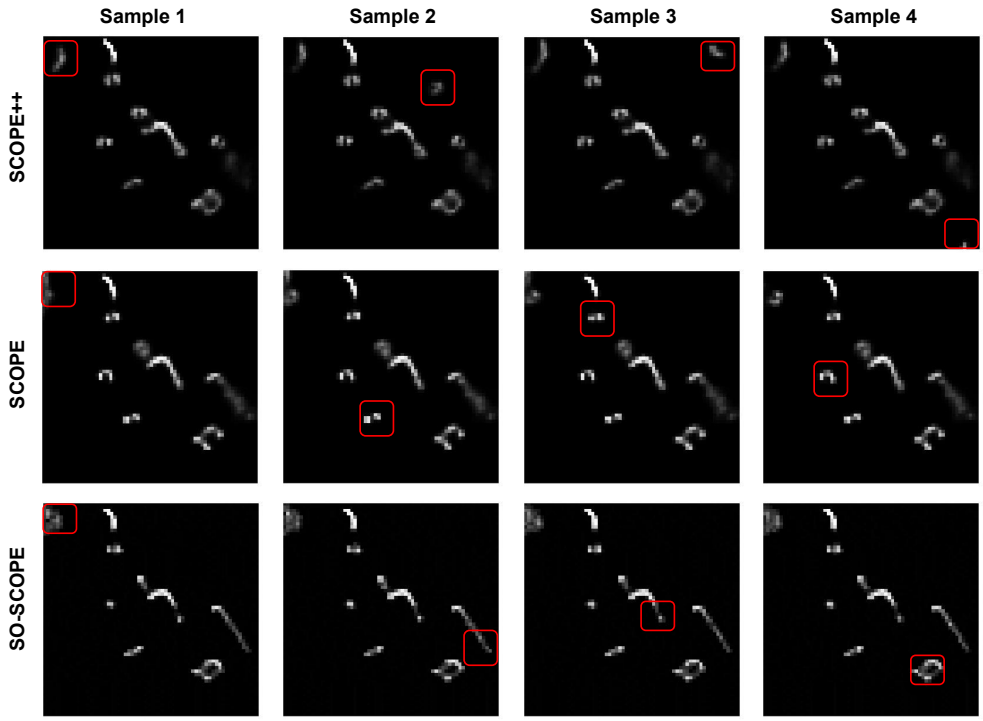
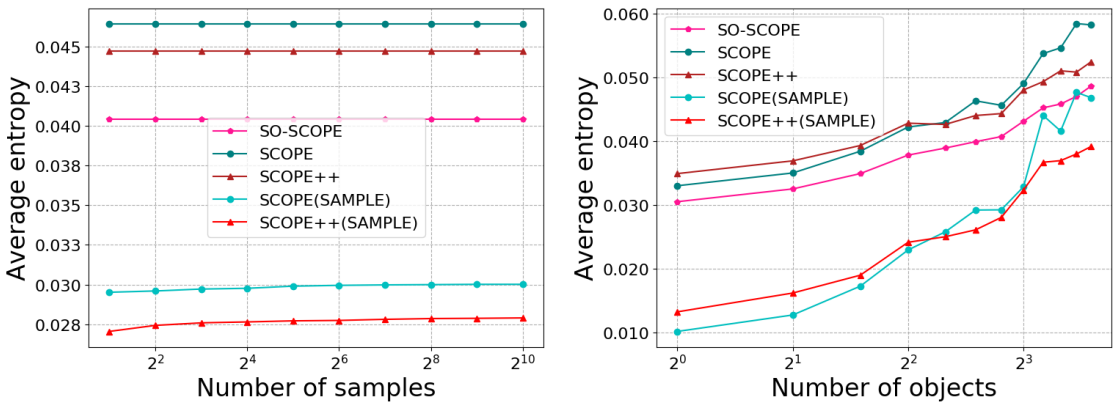



Figure 6.11. A diverse set of prediction samples of our SCOPE++, SCOPE, and SO-SCOPE predictors tested on the OGM-Turtlebot2 dataset at the 5th prediction timestep. The black area is the free space and the white area is the occupied space. The red bounding boxes highlight variations across outputs for the same input.

**Experiment Setup.** Our characterization of the uncertainty has three primary goals. First, to show that the generated OGMs are consistent with reality. Second, to show that the SCOPE family predictors converge to a consistent distribution. Third, to show that the uncertainty in the overall OGM increases with the number of objects in the scene.

For the experiments, we use the OGM-Turtlebot data dataset and fix the prediction horizon to $\tau = 5$. We use the procedure outlined in Fig. 6.5 to count the number of objects in each OGM. Then, we randomly select 20 input sequences for each number of objects (from 1 to 12) and generate a total of 1,024 OGM samples for each input sequence using each SCOPE method.




(a) Entropy vs. Number of samples

(b) Entropy vs. Number of objects

Figure 6.12. Average entropy of our SCOPE, SCOPE++, and SO-SCOPE predictors at 5th prediction time step on OGM-Turtlebot2 dataset.

**Qualitative results.** Our proposed SCOPE++, SCOPE, and SO-SCOPE have similar prediction performance and are able to generate prediction samples. Figure 6.11 showcases a diverse set of representative prediction samples. We see that all three methods show variation across samples and generate realistic-looking structures, such as cylinders for legs and lines for walls. We also see that SCOPE++ shows more variation across samples than SO-SCOPE and SCOPE.

**Quantitative results.** One of the biggest differences between our proposed methods compared to state-of-the-art baselines is that the SCOPE family can provide uncertainty estimates. The goal of these tests is to analyze the output distribution of our SCOPE, SCOPE++, and SO-SCOPE predictors.

**1) Convergence.** First, we show how the entropy of the final probabilistic OGM changes as the number of OGM samples increases, with the hypothesis that it will level off at some value well below that of the entropy of a uniform distribution. Figure 6.12(a) shows that the average entropy of SCOPE(SAMPLE) and SCOPE++(SAMPLE) predictors (*i.e.,*

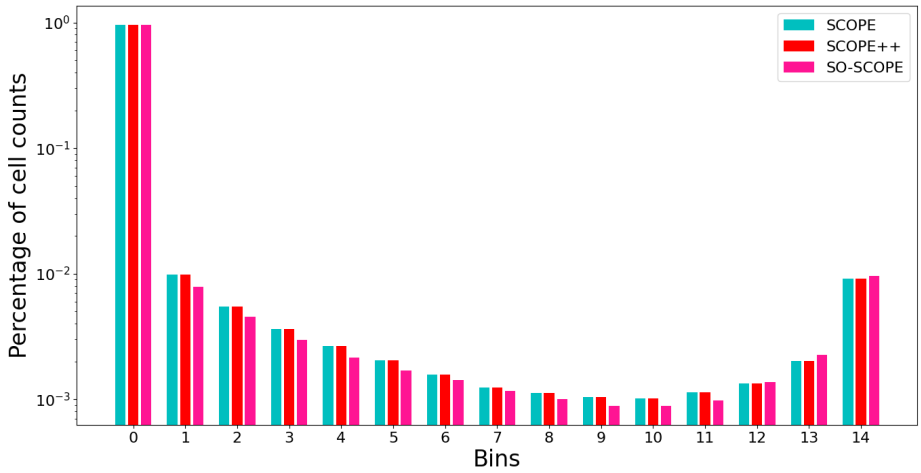



Figure 6.13. Cell counts per bin in the SCOPE, SCOPE++, and SO-SCOPE predictors.

computed via (6.15c)) starts to converge and plateau at a certain value after generating 128 samples, which is consistent with our hypothesis. Note that the entropy values for SCOPE, SCOPE++, and SO-SCOPE are flat lines because we use (6.15b) for calculation, and no sampling is required. Furthermore, the differences among the three flat lines are small (*i.e.,* no more than 0.004), which indicates that our uncertainty quantification for SCOPE++ is correct and can provide reasonable uncertainty estimates for our SO-SCOPE predictor. The differences of them are mainly caused by the differences in their predicted OGMs. Fig. 6.13 shows the differences in cell counts for each bin in their predicted OGMs. It is consistent with the entropy differences of our SCOPE, SCOPE++, and SO-SCOPE predictors, where the difference between SCOPE and SCOPE++ is smaller than the difference between SO-SCOPE and SCOPE++ (*i.e.,* SO-SCOPE has a smaller percentage of cell counts or lower entropy near bin 1 to bin 11).

However, we can see that there is a gap between the entropy calculated by (6.15b) using the mean map $\bar{\mathbf{o}}$ and the entropy calculated by (6.15c) (*i.e.,* SCOPE/SCOPE++ is higher than SCOPE(SAMPLE)/SCOPE++(SAMPLE)). We believe this gap stems from two fac-

tors. First, our assumption that cells $c$ within an OGM are independent will overestimate the uncertainty values because actually knowing the occupancy of one cell should reduce the uncertainty of neighboring cells (and our learned latent representation in the VAE is encoding these correlations). Second, the mean map $\bar{\mathbf{o}}$ is different individual samples $\tilde{\mathbf{o}}$, so the statistics differ slightly.

**2) Scene Complexity.** We also wish to show how the entropy of the final probabilistic OGM changes as the number of objects in the OGM increases, with the hypothesis that it will increase with the number of objects in it. Figure 6.12(b) clearly demonstrates that this is true, with the entropy of SCOPE/SCOPE(SAMPLE), SCOPE++/SCOPE++(SAMPLE) and SO-SCOPE predictors increasing with the number of objects. This confirms the intuition that there is more uncertainty in a scene with more (dynamic) objects in it.

## 6.4 Uncertainty-Aware Navigation

We next test the applicability of our proposed OGM predictors to practical mobile robot navigation in crowded dynamic scenes. While most approaches [5, 6, 8, 71] use the past or current environment states (*i.e.,* raw sensor data and preprocessed data representations) as perception information for their navigation control policies, many other methods [81, 82, 84, 112, 152, 153] believe that predicted future states can help robots better avoid collisions with pedestrians, and start utilizing the predicted environment states (*e.g.,* pedestrian trajectory prediction and latent state prediction) to improve navigation performance through pedestrian crowds. However, most prediction-based navigation policies assume the outcomes of the interactive environments and their control actions are deterministic. Due to the stochastic nature of the robot and its surrounding environments, there

are many uncertainties in environment prediction and control actions. Based on this trend, many works [101, 154–158] consider how to estimate uncertainties in environment prediction, and how to fuse these uncertainties into navigation systems to provide robust risk-aware or uncertainty-aware control policies. However, most of these uncertainty-aware works [154–157] are dedicated to specific control policies (*i.e.,* MPC algorithms) and cannot directly apply their uncertainty information to currently existing model-based and learning-based control policies.

To leverage the uncertainty information in the environmental future states and provide robust and reliable navigation behavior, we propose a general navigation control policy framework (shown in Fig, 6.14) based on costmaps, which are used in the `move_base` ROS navigation framework and can be integrated with most currently exiting control policies. Specifically, we use prediction and uncertainty costmaps to tell the robot the potentially dangerous areas ahead of it and how certain they are, which enables mobile robots to make safer nominal path plans, take proactive actions to avoid potential collisions, and improve navigation capabilities. Note that each costmap grid cell of our proposed prediction and uncertainty costmaps has an initial constant cost, and we map each occupied grid cell of the prediction costmap and uncertainty costmap to a Gaussian obstacle value rather than a "lethal" obstacle value. This is because the predicted obstacles and uncertainty regions are not real obstacle spaces.

The basic framework of our proposed predictive uncertainty-aware navigation planner is shown in Fig. 6.14, and there are two versions: based on original SCOPE and based on optimized SO-SCOPE. The key idea of our proposed framework is that we utilize our SCOPE or SO-SCOPE predictor to generate a prediction map and an uncertainty map,

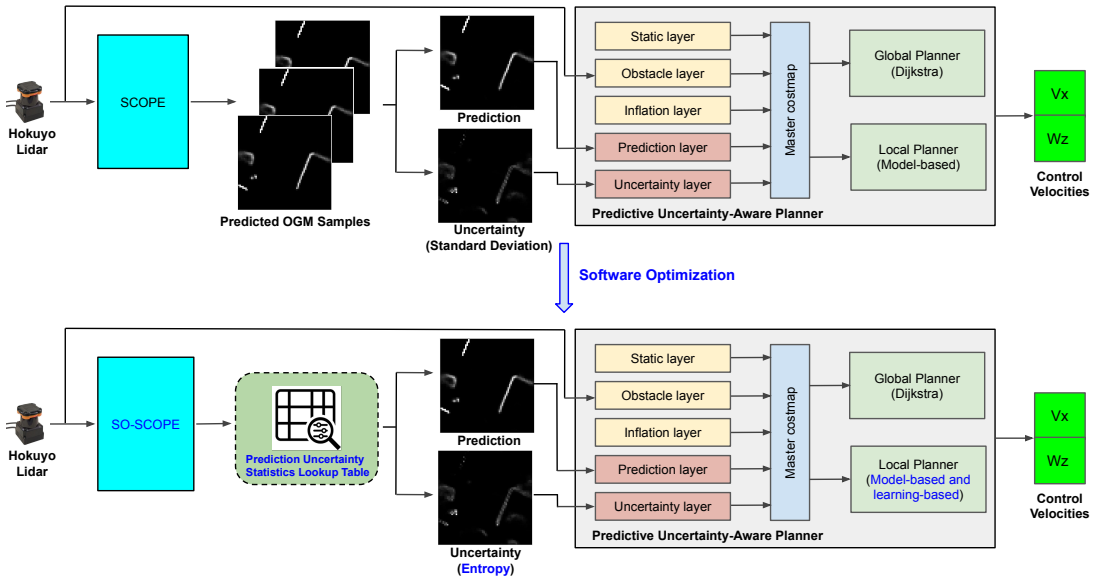



Figure 6.14. System architectures of the proposed SCOPE-based and SO-SCOPE-based predictive uncertainty-aware navigation planners. The blue font emphasizes the difference between the SCOPE-based navigation framework and the SO-SCOPE-based navigation framework. The basic process of our proposed navigation framework are: first, the lidar data is also fed into our SCOPE or SO-SCOPE predictor to generate predicted OGM samples or lookup statistics from the prediction uncertainty statistics lookup table. Then, we can easily generate the prediction mean map and uncertainty map from these samples or the statistics. Finally, we create two new costmap layers, the prediction costmap layer, and the uncertainty costmap layer, combine them into the master costmap, and obtain our SCOPE-based or SO-SCOPE-based predictive uncertainty aware planners.

which are used to generate the prediction costmap layer and uncertainty costmap layer respectively for our predictive uncertainty-aware planner. Note that each costmap grid cell of our proposed prediction and uncertainty costmaps has an initial constant cost, and we map each occupied grid cell of the prediction costmap and uncertainty costmap to a Gaussian obstacle value rather than a "lethal" obstacle value. This is because the predicted obstacles and uncertainty regions are not real obstacle spaces.

There are two versions: based on SCOPE and SO-SCOPE. The difference between SCOPE-based planner and SO-SCOPE-based planner is the method of generating prediction and uncertainty maps. The SCOPE-based planner needs to use the VAE module to

generate prediction samples to provide a prediction map (*i.e.,* mean) and an uncertainty map (*i.e.,* standard deviation), which requires a large number of samples to provide accurate estimates and is time-consuming and memory-consuming. In contrast, our simple SO-SCOPE-based planner only needs to use a simple prediction uncertainty statistics lookup table to provide a prediction map (*i.e.,* SO-SCOPE output) and an uncertainty map (*i.e.,* entropy), which has fast running speed and small memory consumption. Note that the SCOPE-based navigation framework can only run with traditional model-based local planners (*e.g.,* DWA [5] and VO-based planners [59, 60]) in the resource-limited robots while the SO-SCOPE-based navigation framework can run with any types of model-based and learning-based local planners (*e.g.,* CNN [6], A1RC [71], and DRL-VO [8]).

### *6.4.1 Baselines and Evaluation Metrics*

We first use the model-based DWA [5] as the local planner to instantiate a SCOPE-based predictive uncertainty-aware planner (*i.e.,* DWA-SCOPE-PU) to demonstrate how prediction and its uncertainty information improve robot navigation performance. Then, we use both the model-based DWA [5] and the learning-based DRL-VO [8] as the local planners to instantiate two SO-SCOPE-based predictive uncertainty-aware planners (*i.e.,* DWA-SO-SCOPE-PU, DRL-VO-SO-SCOPE-PU) to demonstrate the software-optimized SO-SCOPE predictor is hardware friendly and our proposed SO-SCOPE-based navigation framework can be applied to any currently exiting control policies.

We test these two predictive uncertainty-aware control policies, along with four state-of-the-art control policies: model-based DWA planner [5], supervised-learning-based CNN [6], DRL-based A1-RC [71], and DRL-based DRL-VO [8], and two ablation prediction-

aware control policies without uncertainty costmaps: DWA-DeepTracking-P and DWA-SCOPE-P. Note that we use these four state-of-the-art baseline policies directly from their papers [5, 6, 8, 71] without any retraining or parameter tuning.

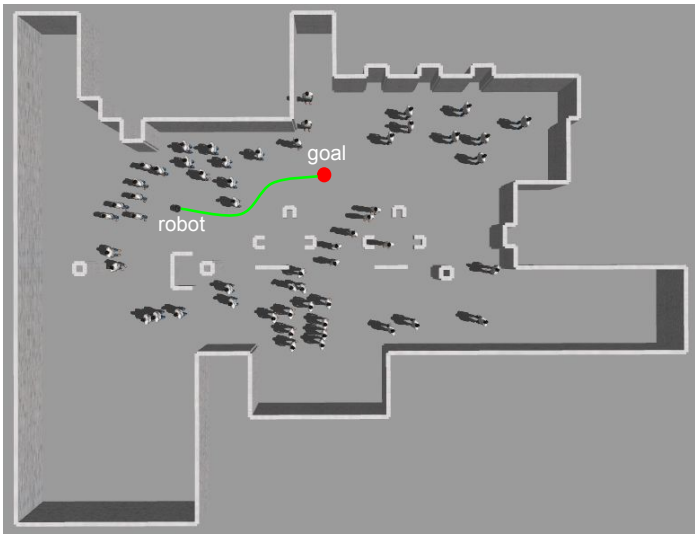
To evaluate the performance of navigation control policies, we use the following four metrics from [6, 8]:

- **Success rate**: the fraction of collision-free trials
- **Average time**: the average travel time of trials
- **Average length**: the average trajectory length of trials
- **Average speed**: the average speed during trials

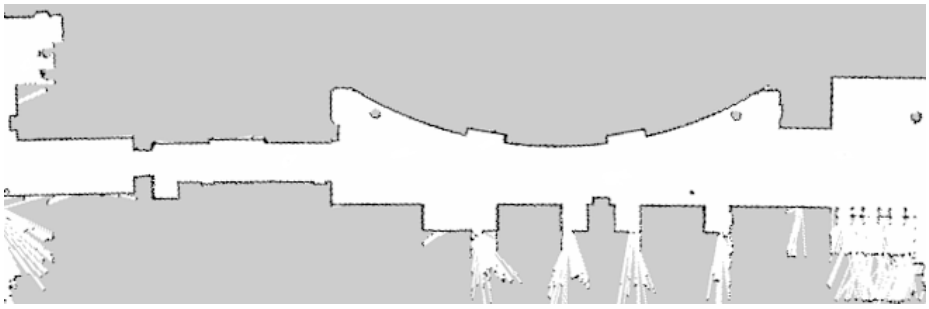
### *6.4.2 Experiment Setup*

Following the navigation evaluation settings in work [8], all control policies are tested by a Turtlebot2 robot with a maximum speed of 0.5 m/s, equipped with a Hokuyo UTM-30LX lidar and a ZED stereo camera, in the simulated Lobby Gazebo world with 15-45 pedestrians (sampling interval 10), as shown in Fig. 6.15(a). The measurement range of Hokuyo lidar is set to $[0.1, 30]$ m, its FOV is $270°$, and its angular resolution is $0.25°$. The depth range of the ZED camera is set to $[0.3, 20]$ m, and its FOV is $90°$.

In addition, we also deploy our proposed DWA-SCOPE-PU and DRL-VO-SO-SCOPE-PU control policies on a real Turtlebot2 robot, which has the same configuration as the simulated robot but uses an NVIDIA Jetson AVG Xavier embedded computer as its main computing device. We conduct testing in an indoor hallway environment at Temple University during peak traffic periods between classes, the floor plan of which is shown in




(a) Gazebo lobby



(b) Indoor hallway

Figure 6.15. Robot navigation test environments in the simulation and real world. The Lobby Gazebo world has 4 crowd density configurations ranging from 15-45 (sampling interval 10) pedestrians.

Fig. 6.15(b). Note that considering the computational resources of the Turtlebot2 robot, we use the SCOPE predictor to only generate 8 predicted OGM samples at the 6th prediction time step (*i.e.,* 0.6 s) for the DWA-SCOPE-PU control policy and it cannot combine with any learning-based control policies. In contrast, our software-optimized SO-SCOPE predictions do not suffer from such problems and can be combined with model-based or learning control strategies (*e.g.,* with DRL-VO in our instantiation).

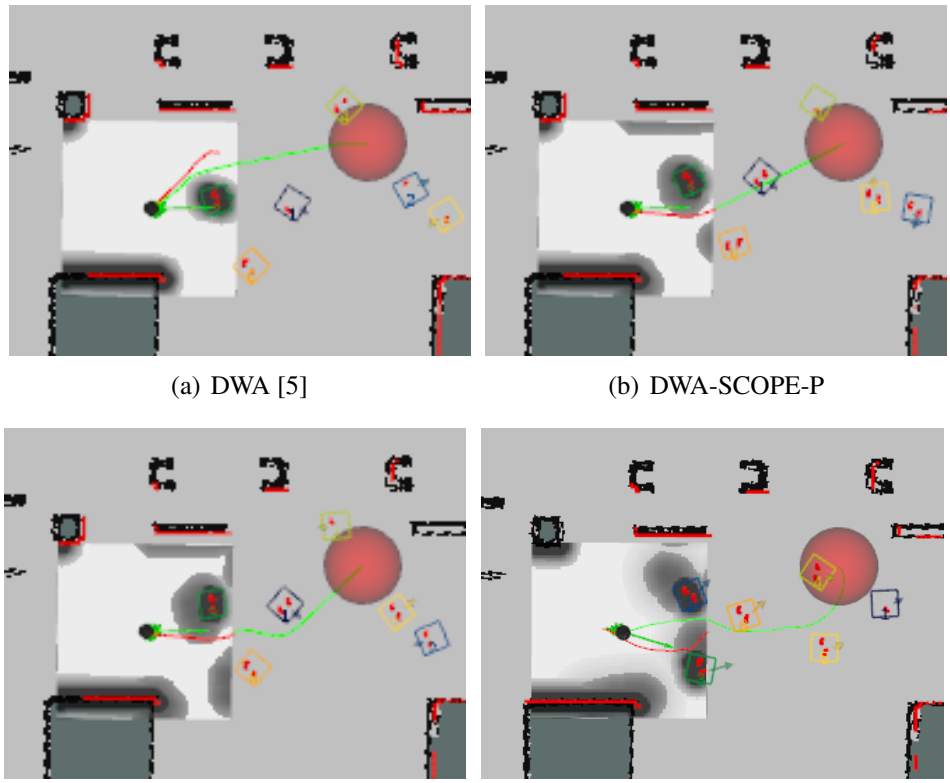
Table 6.3. Navigation results in the Lobby world at different crowd densities.

| Environment | Method | Success Rate | Average Time (s) | Average Length (m) | Average Speed (m/s) |
|---|---|---|---|---|---|
| Lobby world, 15 pedestrians | CNN [6] | - | - | - | - |
| | A1-RC [71] | 0.94 | 14.36 | 6.40 | 0.45 |
| | DWA [5] | 0.94 | 12.21 | 5.09 | 0.42 |
| | DWA-DeepTracking-P | 0.95 | 12.10 | **5.03** | 0.42 |
| | DWA-SCOPE-P | 0.93 | 12.62 | 5.06 | 0.40 |
| | DWA-SCOPE-PU | **0.98** | 12.79 | 5.06 | 0.40 |
| | DWA-SO-SCOPE-P | 0.94 | 12.12 | 5.10 | 0.42 |
| | DWA-SO-SCOPE-PU | 0.96 | 13.25 | 5.12 | 0.39 |
| | DRL-VO [8] | 0.95 | **11.34** | 5.26 | **0.46** |
| | DRL-VO-SO-SCOPE-PU | 0.97 | 11.69 | 5.43 | **0.46** |
| Lobby world, 25 pedestrians | CNN [6] | 0.80 | 19.31 | 6.16 | 0.32 |
| | A1-RC [71] | 0.88 | 14.18 | 6.26 | 0.44 |
| | DWA [5] | 0.82 | 13.49 | 5.12 | 0.38 |
| | DWA-DeepTracking-P | 0.87 | 12.67 | **5.06** | 0.40 |
| | DWA-SCOPE-P | 0.90 | 13.53 | 5.08 | 0.37 |
| | DWA-SCOPE-PU | 0.91 | 13.75 | 5.07 | 0.37 |
| | DWA-SO-SCOPE-P | 0.90 | 13.23 | 5.13 | 0.39 |
| | DWA-SO-SCOPE-PU | 0.92 | 14.26 | 5.19 | 0.36 |
| | DRL-VO [8] | 0.92 | **11.37** | 5.29 | **0.47** |
| | DRL-VO-SO-SCOPE-PU | **0.93** | 11.95 | 5.52 | 0.46 |
| Lobby world, 35 pedestrians | CNN [6] | 0.81 | 14.30 | 5.40 | 0.38 |
| | A1-RC [71] | 0.77 | 16.81 | 6.89 | 0.41 |
| | DWA [5] | 0.82 | 14.18 | 5.15 | 0.36 |
| | DWA-DeepTracking-P | 0.84 | 13.93 | **5.10** | 0.37 |
| | DWA-SCOPE-P | 0.86 | 13.79 | 5.12 | 0.34 |
| | DWA-SCOPE-PU | 0.89 | 14.90 | 5.12 | 0.34 |
| | DWA-SO-SCOPE-P | 0.86 | 14.03 | 5.16 | 0.37 |
| | DWA-SO-SCOPE-PU | 0.88 | 15.53 | 5.20 | 0.33 |
| | DRL-VO [8] | 0.88 | **11.42** | 5.31 | **0.46** |
| | DRL-VO-SO-SCOPE-PU | **0.92** | 12.26 | 5.57 | 0.45 |
| Lobby world, 45 pedestrians | CNN [6] | 0.79 | 16.65 | 5.62 | 0.34 |
| | A1-RC [71] | 0.77 | 14.65 | 6.28 | 0.43 |
| | DWA [5] | 0.77 | 15.39 | 5.16 | 0.34 |
| | DWA-DeepTracking-P | 0.78 | 15.23 | **5.14** | 0.34 |
| | DWA-SCOPE-P | 0.79 | 14.84 | **5.14** | 0.35 |
| | DWA-SCOPE-PU | 0.82 | 15.96 | 5.17 | 0.32 |
| | DWA-SO-SCOPE-P | 0.78 | 14.89 | 5.21 | 0.35 |
| | DWA-SO-SCOPE-PU | 0.80 | 16.29 | 5.27 | 0.32 |
| | DRL-VO [8] | 0.81 | **11.65** | 5.37 | **0.46** |
| | DRL-VO-SO-SCOPE-PU | **0.83** | 12.24 | 5.56 | 0.45 |

### *6.4.3 Simulation Results*

Table 6.3 summarizes these navigation results, where we observe three key phenomena. First, compared to DWA series variants (*i.e.,* DWA, DWA-DeepTracking-P, DWA-SCOPE-P, and DWA-SO-SCOPE-P), our proposed DWA-SCOPE-PU and DWA-SO-SCOPE-PU policies have the highest success rate in each crowd size, while DWA-SCOPE-PU has almost the shortest path length. This shows that the prediction costmap from our OGM predictors is able to help the currently existing model-based navigation policy (*i.e.,* DWA) to provide safer and shorter paths, and combining it with its associated uncertainty costmap can achieve a much better navigation performance (*i.e.,* PU is better than P alone). Furthermore, it shows that our software-optimized SO-SCOPE predictor can maintain good robot navigation performance even though it leads to longer paths compared to the SCOPE predictor. All these results demonstrate the effectiveness of our proposed SCOPE series predictors in helping design safe robot navigation policies in crowded dynamic scenes.

The reasons why the OGM prediction and its uncertainty information can improve robot safe navigation can be explained qualitatively through Fig. 6.16. It shows the difference of nominal paths and costmaps generated by four different planners (*i.e.,* DWA, DWA-SCOPE-P, DWA-SCOPE-PU, and DWA-SO-SCOPE-PU) in the simulated lobby environment. The default DWA planner [5] only cares about the current state of the environment and generates a costmap based on the perceived obstacles. The predictive DWA planner (*i.e.,* DWA-SCOPE-P) using the prediction map of our proposed SCOPE predictor can generate a costmap with predicted obstacles. The predictive uncertainty-aware DWA planner (*i.e.,* DWA-SCOPE-PU and DWA-SO-SCOPE-PU) using both the predic-



(a) DWA [5]

(b) DWA-SCOPE-P

(c) DWA-SCOPE-PU

(d) DWA-SO-SCOPE-PU

Figure 6.16. Robot reactions and their corresponding costmaps generated by different DWA-based control policies in the simulated lobby environment. The robot (black disk) is avoiding pedestrians (colorful square boxes) and reaching the goal (red disk) according to the nominal path (green line) planned by the costmap (square white map).

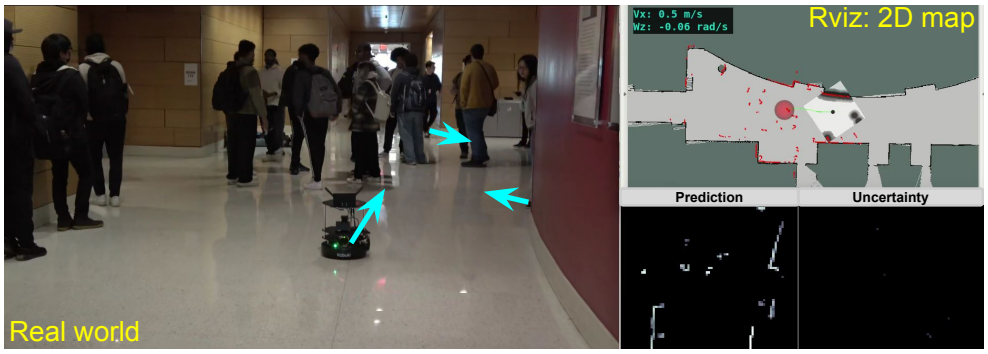
tion map and uncertainty map of our proposed SCOPE predictor can generate a safer costmap with predicted obstacles and uncertainty regions. Note that the path generated by DWA-SO-SCOPE-PU is more tortuous than that of DWA-SCOPE-PU, which may explain why DWA-SO-SCOPE-PU requires longer paths to achieve a similar success rate as DWA-SCOPE-PU. These additional predicted obstacles and uncertainty regions of our proposed DWA-SCOPE-PU and DWA-SO-SCOPE-PU planners enable the robot to follow safer nominal paths and reduce collisions with obstacles, especially moving pedestrians. See the accompanying Multimedia for a detailed simulation navigation demonstration.

Second, our proposed DWA-SCOPE-P and DWA-SO-SCOPE-P have a higher success rate than DWA-DeepTracking-P at every crowd size, indicating that higher prediction accuracy can also improve safe navigation performance. Third, compared with all other state-of-the-art control policies, our proposed DRL-VO-SO-SCOPE-PU policy has the highest success rate and the fastest average speed in every situation. This suggests that our software-optimized SO-SCOPE predictor using a simple knowledge distillation network and prediction uncertainty statistical lookup table can also provide additional benefits for the learning-based safe navigation performance in crowded dynamic environments with varying crowd densities. Furthermore, it shows the potential capability of our software-optimized predictive uncertainty-aware navigation framework for different high computational load learning-based control policies.

### *6.4.4 Hardware Results*

Besides the simulated experiments, we also conduct a real-world experiment to demonstrate the applicability of our DWA-SCOPE-PU and DRL-VO-SO-SCOPE-PU policies in the real world. Both control policies use the `amcl` ROS package to provide the robot localization service in known maps. Note that our DRL-VO-SO-SCOPE-PU control policy requires additional running of three deep learning networks (*i.e.,* YOLOv3 [115], SO-SCOPE, and DRL-VO) and one optimization-based multiple hypothesis tracker (MHT) [116] to implement corresponding pedestrian detection, environment OGM prediction, navigation control, and pedestrian tracking.

From the attached Multimedia and Fig. 6.17, we can see how our robot deployed with DWA-SCOPE-PU is able to actively avoid collisions with walking students crossing the



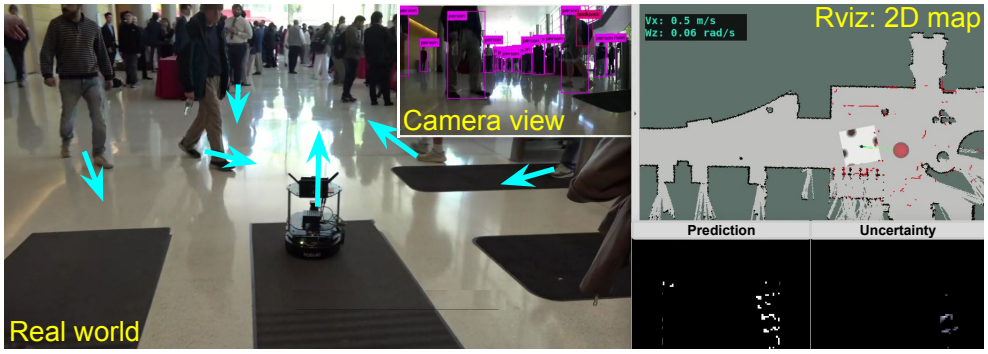

(a) t

(b) t+3

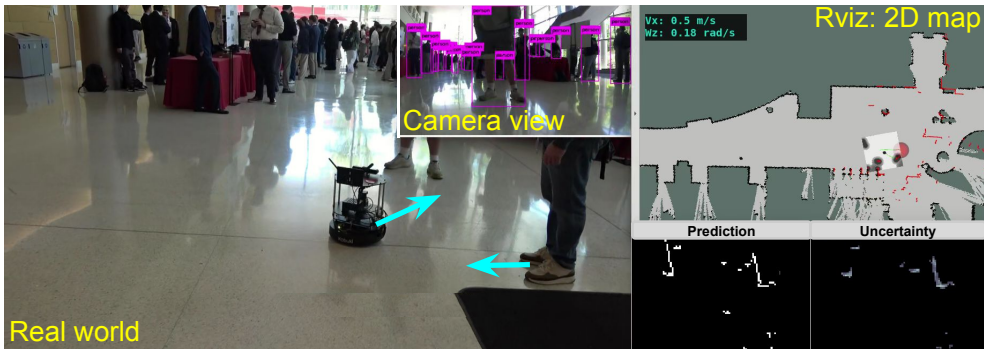
Figure 6.17. Robot deployed with DWA-SCOPE-PU reactions to moving pedestrians in the indoor hallway with the high crowd density at different times.

hallway, safely avoid static students standing in the hallway, and reach predefined goals by following predictive uncertainty-ware nominal paths, traveling a total length of 76.10 m and an average speed of 0.42 m/s. It demonstrates the real-world effectiveness of our proposed SCOPE predictors and DWA-SCOPE-PU planner.

In addition, from the attached Multimedia and Fig. 6.18, we can see that even with three learning-based blocks, our robot deployed with DRL-VO-SO-SCOPE-PU is still able to quickly and actively avoid collisions with walking students crossing the hallway and reach predefined goals by following predictive uncertainty-ware nominal paths, traveling a total length of 86.41 m and an average speed of 0.47 m/s. It demonstrates our software-optimized SO-SCOPE predictor is hardware friendly and our software-optimized predictive




(a) t




(b) t+3

Figure 6.18. Robot deployed with DRL-VO-SO-SCOPE-PU reactions to moving pedestrians in the indoor hallway with the high crowd density at different times.

uncertainty-aware navigation framework can be combined with different high computational load learning-based algorithms.

## 6.5 Conclusion

In this chapter, we propose a family of hardware-friendly stochastic occupancy grid map prediction algorithms (*i.e.,* SCOPE++, SCOPE, and SO-SCOPE) that provides mobile robots with the ability to accurately and robustly predict the future states of complex, dynamic, human-occupied environments. Specifically, we first propose two VAE-based stochastic predictors (*i.e.,* SCOPE++ and SCOPE) that exploit information from robot motion, object motion, and static scene geometry, and use a VAE-based network to fuse this useful information into the latent space and predict the distribution of future environ-

ment states. We then perform software optimization on them using knowledge distillation and uncertainty quantification operations to propose a hardware-friendly SO-SCOPE to extend their practical applications in resource-constrained robots. Furthermore, we propose a novel predictive uncertainty-aware navigation framework by combining these proposed stochastic predictors with the costmap-based ROS navigation stack to improve the performance of current state-of-the-art model-based and learning-based control policies for safe navigation in crowded dynamic scenes.

We demonstrate that our proposed SCOPE family achieves smaller absolute error, higher structure similarity, and higher tracking accuracy than the other state-of-the-art image-based predictors on three different simulated and real-world datasets collected by three different robot models. It also provides a range of plausible and diverse future prediction states for complex stochastic environments. We further demonstrate through operational analysis and experiments on embedded computing devices that our proposed software-optimized SO-SCOPE prediction engine is faster and more memory-efficient, allowing it to provide real-time inference in resource-constrained robots. Lastly, we demonstrate through simulated and hardware experiments that we can easily integrate the predicted maps into the existing navigation framework, all of which benefit from leveraging the prediction and uncertainty information of our SCOPE family. In summary, our SCOPE predictors are hardware-friendly and improve mobile robots' ability to safely navigate through crowded dynamic environments.

# CHAPTER 7
# CONCLUSION

## 7.1 Contributions

In this dissertation, we have proposed different deep learning algorithms to develop learning-based navigation systems that enable a mobile robot to autonomously, efficiently, and reliably navigate in crowded dynamic scenes filled with static obstacles (such as walls and tables), as well as dynamic obstacles (such as mobile robots and dense pedestrians).

After the detailed background material and literature review analysis in Chapter 2 and Chapter 3, we first proposed a new set of preprocessed environmental data representations for modeling static scene geometry and dynamic pedestrians in Chapter 4, which are specialized for crowded dynamic environments and can bridge the appearance gap between imperfect simulation and reality. Specifically, we used a short history of lidar data (*i.e.,* downsampled via minimum and average pooling), the current kinematics of pedestrians (*i.e.,* position and velocity stored in the two grids), and a sub-goal point to model and represent the state of the crowded dynamic environment. The effectiveness of our proposed preprocessed data representations is demonstrated in different simulation experiments with a novel imitation learning-based control policy [6].

Then, leveraging our proposed novel environmental data representations, we proposed a novel DRL-based crowd-aware control policy (*i.e.,* DRL-VO) that is highly generalizable to different crowd densities and unseen contexts in crowded dynamic environments by designing a novel VO-based reward function that can balance collision avoidance and speed in Chapter 5. Specifically, we proposed a novel reward function that uses first-order velocity obstacles to penalize headings that are likely to lead to collisions, thereby guiding

the robot to continuously tune its heading direction to actively avoid crowds of pedestrians while making progress towards the goal. The effectiveness of our proposed DRL-VO control policy is demonstrated in different simulated and real-world navigation experiments with different crowd densities and unseen environments [8]. Note that, this control policy also competed in the benchmark autonomous robot navigation (BARN) Challenge at the 2022 IEEE International Conference on Robotics and Automation (ICRA), where it placed first in the simulation competition and third in the hardware competition [7].

Finally, we proposed a family of hardware-friendly stochastic occupancy grid map prediction algorithms (*i.e.,* SCOPE++, SCOPE, and SO-SCOPE) that enable mobile robots to accurately and robustly predict future states of complex and human-occupied environments, as well as a novel predictive uncertainty-aware navigation framework (*e.g.,* DWA-SCOPE-PU, DWA-SO-SCOPE-PU, and DRL-VO-SO-SCOPE-PU) to improve the performance of current exiting control policies in crowded dynamic scenes in Chapter 6. Specifically, we first proposed two VAE-based stochastic predictors (*i.e.,* SCOPE++ and SCOPE) that exploit information from robot motion, object motion, and static scene geometry, and use a VAE-based network to fuse this useful information into the latent space and predict the distribution of future environment states. We then performed software optimization on them using knowledge distillation and uncertainty quantification operations to propose a hardware-friendly SO-SCOPE to extend their practical applications in resource-constrained robots. Afterward, combining these proposed stochastic predictors with the costmap-based ROS navigation stack, we integrated currently existing model-based and learning-based control policies into our predictive uncertainty-aware navigation framework to improve their navigation performance and provide reliable navigation behaviors. The effectiveness

of our proposed prediction and navigation algorithms is demonstrated on three different datasets collected on different robot models, as well as in simulated and real-world robot navigation experiments [9, 10].

## 7.2 Future Work

While it is encouraging to address some key issues in environmental modeling and representation, environmental prediction, and design of crowd-aware and uncertainty-aware navigation policies( [6–10]) is encouraging, this dissertation's research only focuses on the problems and navigation performance on the robotic side, while ignoring higher-level human comfort. In addition, the basic robot navigation framework used in this paper (as shown in Figure 1.2) is also a complex cascade framework, which accumulates the errors of each module and limits the navigation performance. In order to solve these two key problems and promote the development of mobile robots for last-mile delivery, our future work will focus on designing socially-aware control policies and developing a novel lightweight unified navigation framework for mobile robots.

### *7.2.1 Socially-Aware Control Policy*

In addition to autonomous robot navigation methods [5, 59–61, 68–72, 74–77, 83, 85, 103, 104] that focus on collision avoidance, efficiency, and autonomy, many other model-based approaches [67, 159, 160] and learning-based polices [78–80, 110, 111] aim to propose socially-aware control policies and improve the comfort of humans. They all attempt to elucidate complex human-robot interactions (both at the individual and group level) and underlying social rules by learning human navigation behavior from psychological concepts. However, unraveling these complex human-robot interactions and applying explicit

and implicit social norms to robot control strategies remains an open problem. To address this issue, we will propose a novel socially-aware control policy by quantifying factors in complex human-robot interactions and using learning-based networks to learn underlying social norms. Specifically, we first plan to design some human-human and human-robot interaction experiments from a psychological perspective to study human reactions to the robot and quantify some interaction factors. Second, we plan to combine the theory of the social force model and machine learning algorithms to process these interaction factors and capture some underlying social norms between humans and robots. Finally, we plan to transfer these implicit social rules into robot control policies, enabling robots to socially and naturally move around pedestrians.

### *7.2.2 A Lightweight Unified Framework For Robot Navigation*

As shown in Fig. 1.2, the basic robot navigation system is a complex cascade framework consisting of perception, localization, prediction, modeling, planning, and control modules. Researchers in different fields only study and optimize one or two different modules to provide corresponding functions, tacitly assuming that other modules are optimized and provided by others. However, this mainstream cascading framework has two main disadvantages: 1) each module has noise and errors, which cascade and accumulate to the last module, affecting the final navigation performance; 2) it is difficult to optimally schedule the limited computing resources in the robot for each navigation module, especially those using off-the-shelf solutions. To address these issues, many learning-based researchers advocate proposing a fully end-to-end navigation framework that directly uses a deep neural network to map the input sensor data to final control commands. However,

such a fully end-to-end framework only focuses on a single control task and has limited generalization abilities to new environments or new application scenarios. With the help of large language models (LLMs), recent researchers have proposed a unified autonomous driving framework based on LLMs to jointly learn multiple navigation tasks to improve the generalization ability of the final unified control policy. However, such an LLM-based unified framework requires huge computing resources and cannot be directly deployed on mobile robots with limited resources. I aim to propose a lightweight unified neural network framework for mobile robot navigation, which is able to eliminate cascading errors from the cascade framework, has strong generalizability to different robot configurations and different unseen environments, and can run in real-time on resource-limited mobile robots. Specifically, I will first design generic preprocessed intermediate data representations (*e.g.,* graph data) to abstract away the underlying different hardware sensors and model the state of the environment. Second, I plan to unify the perception, estimation, planning, and control modules by implementing multiple tasks with a lightweight neural network (*e.g.,* a graph neural network). Finally, I will combine uncertainty information and human comfort factors and use joint supervised learning and reinforcement learning to hybridly train this unified neural network.

## 7.3 Concluding Remarks

This dissertation proposes four learning-based solutions, from environment modeling and prediction to different control policy designs, to improve the safety and efficiency of robot navigation performance and promote the development of mobile robot navigation in crowded dynamic scenes. Future work will further focus on human comfort aspects,

improve social awareness of robot navigation, and rebuild a new lightweight unified navigation framework. We believe that all the completed work and future work presented in this dissertation will contribute to the development of mobile robot navigation in crowded dynamic scenes, help robots and humans live in a friendly manner, and further make our daily lives more convenient, efficient, and comfortable.

## REFERENCES


[1] H. Carsten and B. Susanne, "Mobile robot transports sterile goods in hospital," https://ifr.org/ifr-press-releases/news/mobile-robot-transports-sterile-goods-in-hospital, Aug 2020, (Accessed on 12/24/2022).

[2] Y. Amer, "Pandemic staffing challenges may be spurring the robot revolution," https://www.wbur.org/news/2022/08/25/robotics-automation-workforce, Aug 2022, (Accessed on 12/24/2022).

[3] J. Paden, "Supermarket chain lucky uses robots to deliver groceries," https://www.amadorvalleytoday.org/20267/news/lucky-uses-robots-to-deliver-groceries/, Mar 2022, (Accessed on 06/12/2024).

[4] E. Lindquist, "This restaurant server has wheels, sensors and an unnatural appeal to customers," https://www.jsonline.com/story/life/food/2022/01/13/robot-restaurant-server-delivering-meals-tables-quick-fun/9036080002/, Jan 2022, (Accessed on 12/24/2022).

[5] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[6] Z. Xie, P. Xin, and P. Dames, "Towards safe navigation through crowded dynamic environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sep. 2021. [Online]. Available: https://doi.org/10.1109/IROS51168.2021.9636102

[7] X. Xiao, Z. Xu, Z. Wang, Y. Song, G. Warnell, P. Stone, T. Zhang, S. Ravi, G. Wang, H. Karnan, J. Biswas, N. Mohammad, L. Bramblett, R. Peddi, N. Bezzo, Z. Xie, and P. Dames, "Autonomous ground navigation in highly constrained spaces: Lessons learned from the Benchmark Autonomous Robot Navigation Challenge at ICRA 2022," *IEEE Robotics & Automation Magazine*, vol. 29, no. 4, pp. 148–156, 2022. [Online]. Available: https://doi.org/10.1109/MRA.2022.3213466

[8] Z. Xie and P. Dames, "DRL-VO: Learning to navigate through crowded dynamic scenes using velocity obstacles," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 2700–2719, Aug. 2023. [Online]. Available: https://doi.org/10.1109/TRO.2023.3257549

[9] ——, "Stochastic occupancy grid map prediction in dynamic scenes," in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine

Learning Research, vol. 229. PMLR, 06–09 Nov 2023, pp. 1686–1705. [Online]. Available: https://proceedings.mlr.press/v229/xie23a.html

[10] ——, "Scope: Stochastic cartographic occupancy prediction engine for uncertainty-aware dynamic navigation," *arXiv preprint arXiv:2407.00144*, 2024.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986," *Biometrika*, vol. 71, pp. 599–607, 1986.

[13] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[19] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[20] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li *et al.*, "Imagination-augmented agents for deep reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[21] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[22] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," *arXiv preprint arXiv:1803.00101*, 2018.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[25] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[29] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[30] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[31] R. Mishra, H. P. Gupta, and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," *arXiv preprint arXiv:2010.03954*, 2020.

[32] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

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

[34] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *Advances in neural information processing systems*, vol. 27, 2014.

[35] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[36] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 2498–2507.

[37] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 45, no. 2, pp. 27–40, 2017.

[38] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[39] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.

[40] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[41] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[42] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-

arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[43] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[44] P. Georgiev, S. Bhattacharya, N. D. Lane, and C. Mascolo, "Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 1–19, 2017.

[45] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[46] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5191–5198.

[47] W. Son, J. Na, J. Choi, and W. Hwang, "Densely guided knowledge distillation using multiple teacher assistants," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9395–9404.

[48] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4794–4802.

[49] J. Yang, H. Zou, S. Cao, Z. Chen, and L. Xie, "Mobileda: Toward edge-domain adaptation," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6909–6918, 2020.

[50] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[52] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[53] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[54] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[55] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, pp. 287–290.

[56] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[57] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[58] B. HomChaudhuri, L. Smith, L. Tapia *et al.*, "Safety, challenges, and performance of motion planners in dynamic environments," in *Robotics research*. Springer, 2020, pp. 793–808.

[59] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[60] D. Wilkie, J. Van Den Berg, and D. Manocha, "Generalized velocity obstacles," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 5573–5578.

[61] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.

[62] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[63] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, pp. 1–29, 2022.

[64] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[66] A. M. Flynn, "Combining sonar and infrared sensors for mobile robot navigation," *The International Journal of Robotics Research*, vol. 7, no. 6, pp. 5–14, 1988.

[67] 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*. IEEE, 2013, pp. 1688–1694.

[68] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 1527–1533.

[69] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. 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.

[70] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 856–892, 2020.

[71] R. Guldenring, M. Görner, N. Hendrich, N. J. Jacobsen, and J. Zhang, "Learning local planners for human-aware navigation in indoor environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6053–6060.

[72] C. Pérez-D'Arpino, C. Liu, P. Goebel, R. Martín-Martín, and S. Savarese, "Robot navigation in constrained pedestrian environments using reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[73] A. Pokle, R. Martín-Martín, P. Goebel, V. Chow, H. M. Ewald, J. Yang, Z. Wang, A. Sadeghian, D. Sadigh, S. Savarese *et al.*, "Deep local trajectory replanning and control for robot navigation," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 5815–5822.

[74] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2759–2764.

[75] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "DroNet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[76] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[77] S. Liu, P. Chang, W. Liang, N. Chakraborty, and K. Driggs-Campbell, "Decentralized structural-rnn for robot crowd navigation with deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[78] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.

[79] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[80] ——, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021.

[81] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva, "Relational graph learning for crowd navigation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 007–10 013.

[82] S. Liu, P. Chang, Z. Huang, N. Chakraborty, K. Hong, W. Liang, D. L. McPherson, J. Geng, and K. Driggs-Campbell, "Intention aware robot crowd navigation with attention-based interaction graph," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 015–12 021.

[83] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[84] A. J. Sathyamoorthy, J. Liang, U. Patel, T. Guan, R. Chandra, and D. Manocha, "Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 345–11 352.

[85] X. Huang, H. Deng, W. Zhang, R. Song, and Y. Li, "Towards multi-modal perception-based navigation: A deep reinforcement learning method," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4986–4993, 2021.

[86] A. Ess, K. Schindler, B. Leibe, and L. Van Gool, "Object detection and tracking for autonomous navigation in dynamic environments," *The International Journal of Robotics Research*, vol. 29, no. 14, pp. 1707–1725, 2010.

[87] D. Nuss, S. Reuter, M. Thom, T. Yuan, G. Krehl, M. Maile, A. Gern, and K. Dietmayer, "A random finite set approach for dynamic occupancy grid maps with real-time application," *The International Journal of Robotics Research*, vol. 37, no. 8, pp. 841–866, 2018.

[88] P. Ondruska and I. Posner, "Deep tracking: Seeing beyond seeing using recurrent neural networks," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[89] M. Itkina, K. Driggs-Campbell, and M. J. Kochenderfer, "Dynamic environment prediction in urban scenes using recurrent representation learning," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2052–2059.

[90] M. Toyungyernsub, M. Itkina, R. Senanayake, and M. J. Kochenderfer, "Double-prong convlstm for spatiotemporal occupancy prediction in dynamic environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 13 931–13 937.

[91] M. Schreiber, S. Hoermann, and K. Dietmayer, "Long-term occupancy grid prediction using recurrent neural networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9299–9305.

[92] M. Schreiber, V. Belagiannis, C. Gläser, and K. Dietmayer, "Motion estimation in occupancy grid maps in stationary settings using recurrent neural networks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8587–8593.

[93] B. Lange, M. Itkina, and M. J. Kochenderfer, "Attention augmented convlstm for environment prediction," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1346–1353.

[94] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[95] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," in *International Conference on Learning Representations*, 2016.

[96] V. L. Guen and N. Thome, "Disentangling physical dynamics from unknown factors for unsupervised video prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 474–11 484.

[97] M. Schreiber, V. Belagiannis, C. Gläser, and K. Dietmayer, "Dynamic occupancy grid mapping with recurrent neural networks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6717–6724.

[98] J. Dequaire, P. Ondrúška, D. Rao, D. Wang, and I. Posner, "Deep tracking in the wild: End-to-end tracking using recurrent neural networks," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 492–512, 2018.

[99] Y. Song, Y. Tian, G. Wang, and M. Li, "2d lidar map prediction via estimating motion flow with gru," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6617–6623.

[100] H. Thomas, M. G. de Saint Aurin, J. Zhang, and T. D. Barfoot, "Learning spatiotemporal occupancy grid maps for lifelong navigation in dynamic scenes," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 484–490.

[101] H. Thomas, J. Zhang, and T. D. Barfoot, "The foreseeable future: Self-supervised learning to predict dynamic scenes for indoor navigation," *IEEE Transactions on Robotics*, 2023.

[102] N. Mohajerin and M. Rohani, "Multi-step prediction of occupancy grid maps with recurrent neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 600–10 608.

[103] P. Singamaneni, A. Favier, and R. Alami, "Human-aware navigation planner for diverse human-robot contexts," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sep. 2021.

[104] S. H. Arul and D. Manocha, "V-rvo: Decentralized multi-agent collision avoidance using voronoi diagrams and reciprocal velocity obstacles," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sep. 2021.

[105] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4459–4466, 2019.

[106] X. Shen, E. L. Zhu, Y. R. Stürz, and F. Borrelli, "Collision avoidance in tightly-constrained environments without coordination: a hierarchical control approach," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[107] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, p. 4282, 1995.

[108] M. Sebastian, S. B. Banisetty, and D. Feil-Seifer, "Socially-aware navigation planner using models of human-human interaction," in *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, pp. 405–410.

[109] M. Boldrer, M. Andreetto, S. Divan, L. Palopoli, and D. Fontanelli, "Socially-aware reactive obstacle avoidance strategy based on limit cycle," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3251–3258, 2020.

[110] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6015–6022.

[111] Y. Chen, C. Liu, B. E. Shi, and M. Liu, "Robot navigation in crowds by graph convolutional networks with attention learned from human gaze," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2754–2761, 2020.

[112] D. Dugas, J. Nieto, R. Siegwart, and J. J. Chung, "Navrep: Unsupervised representations for reinforcement learning of robot navigation in dynamic human environments," *arXiv preprint arXiv:2012.04406*, 2020.

[113] P. Xu and I. Karamouzas, "Human-inspired multi-agent navigation using knowledge distillation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[114] U. Patel, N. K. S. Kumar, A. J. Sathyamoorthy, and D. Manocha, "Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[115] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[116] K. Yoon, Y.-m. Song, and M. Jeon, "Multiple hypothesis tracking algorithm for multi-target multi-camera tracking with disjoint views," *IET Image Processing*, vol. 12, no. 7, pp. 1175–1184, 2018.

[117] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2020.

[118] S. S. Sohn, H. Zhou, S. Moon, S. Yoon, V. Pavlovic, and M. Kapadia, "Laying the foundations of deep long-term crowd flow prediction," in *European Conference on Computer Vision*. Springer, 2020, pp. 711–728.

[119] N. Bellotto and H. Hu, "Multisensor-based human detection and tracking for mobile service robots," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 1, pp. 167–181, 2008.

[120] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, "Multiple hypothesis tracking revisited," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4696–4704.

[121] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll, "What the constant velocity model can teach us about pedestrian motion prediction," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1696–1703, 2020.

[122] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, 2018.

[123] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[124] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.

[125] C. Gloor, "PEDSIM: Pedestrian crowd simulation," *URL http://pedsim.silmaril.org*, vol. 5, no. 1, 2016.

[126] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8026–8037.

[127] A. Gosavi, "Reinforcement learning: A tutorial survey and recent advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, 2009.

[128] M. Moussaïd, D. Helbing, S. Garnier, A. Johansson, M. Combe, and G. Theraulaz, "Experimental study of the behavioural mechanisms underlying self-organization in human crowds," *Proceedings of the Royal Society B: Biological Sciences*, vol. 276, no. 1668, pp. 2755–2762, 2009.

[129] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," https://github.com/DLR-RM/stable-baselines3, 2019.

[130] Y. Chen, H. Peng, and J. Grizzle, "Obstacle avoidance for low-speed autonomous vehicles with barrier function," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 1, pp. 194–206, 2017.

[131] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2020, pp. 116–121.

[132] Z. Wang, X. Xiao, A. J. Nettekoven, K. Umasankar, A. Singh, S. Bommakanti, U. Topcu, and P. Stone, "From agile ground to aerial navigation: Learning from learned hallucination," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 148–153.

[133] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 802–807.

[134] Z. Xu, B. Liu, X. Xiao, A. Nair, and P. Stone, "Benchmarking reinforcement learning techniques for autonomous navigation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9224–9230.

[135] Z. Xu, G. Dhamankar, A. Nair, X. Xiao, G. Warnell, B. Liu, Z. Wang, and P. Stone, "Applr: Adaptive planner parameter learning from reinforcement," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[136] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[137] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[138] K. Katyal, K. Popek, C. Paxton, P. Burlina, and G. D. Hager, "Uncertainty-aware occupancy map prediction using generative networks for robot navigation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5453–5459.

[139] N. L. Baisa, "Derivation of a constant velocity motion model for visual tracking," *arXiv preprint arXiv:2005.00844*, 2020.

[140] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Autonomous robots*, vol. 15, no. 2, pp. 111–127, 2003.

[141] S. Stanton, P. Izmailov, P. Kirichenko, A. A. Alemi, and A. G. Wilson, "Does knowledge distillation really work?" *Advances in Neural Information Processing Systems*, vol. 34, pp. 6906–6919, 2021.

[142] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, "Knowledge distillation: A good teacher is patient and consistent," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 925–10 934.

[143] E. Beale, "Confidence regions in non-linear estimation," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 22, no. 1, pp. 41–76, 1960.

[144] H. Karnan, A. Nair, X. Xiao, G. Warnell, S. Pirk, A. Toshev, J. Hart, J. Biswas, and P. Stone, "Socially compliant navigation dataset (scand): A large-scale dataset of demonstrations for social navigation," *arXiv preprint arXiv:2203.15041*, 2022.

[145] Z. Xie and P. Dames, "Stochastic Occupancy Grid Map Prediction in Dynamic Scenes: Dataset," https://doi.org/10.5281/zenodo.7051560.

[146] N. Ponomarenko, S. Krivenko, K. Egiazarian, V. Lukin, and J. Astola, "Weighted mean square error for estimation of visual quality of image denoising methods," in *CD ROM Proceedings of VPQM*, vol. 5. Scottsdale USA, 2010.

[147] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[148] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3447–3457, 2008.

[149] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2650–2658.

[150] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[151] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.

[152] K. Li, M. Shan, K. Narula, S. Worrall, and E. Nebot, "Socially aware crowd navigation with multimodal pedestrian trajectory prediction for autonomous vehicles," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8.

[153] K. D. Katyal, G. D. Hager, and C.-M. Huang, "Intent-aware pedestrian prediction for adaptive crowd navigation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3277–3283.

[154] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[155] B. Lütjens, M. Everett, and J. P. How, "Safe reinforcement learning with model uncertainty estimates," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8662–8668.

[156] X. Tang, K. Yang, H. Wang, J. Wu, Y. Qin, W. Yu, and D. Cao, "Prediction-uncertainty-aware decision-making for autonomous vehicles," *IEEE Transactions on Intelligent Vehicles*, 2022.

[157] S. Sekiguchi, A. Yorozu, K. Kuno, M. Okada, Y. Watanabe, and M. Takahashi, "Uncertainty-aware non-linear model predictive control for human-following companion robot," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8316–8322.

[158] G. Georgakis, B. Bucher, A. Arapin, K. Schmeckpeper, N. Matni, and K. Daniilidis, "Uncertainty-driven planner for exploration and navigation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 11 295–11 302.

[159] X.-T. Truong and T. D. Ngo, "Toward socially aware robot navigation in dynamic and crowded environments: A proactive social motion model," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 4, pp. 1743–1760, 2017.

[160] G. Ferrer, A. G. Zulueta, F. H. Cotarelo, and A. Sanfeliu, "Robot social-aware navigation framework to accompany people walking side-by-side," *Autonomous robots*, vol. 41, no. 4, pp. 775–793, 2017.