

ABSTRACT

Title of Dissertation: **AUTONOMOUS ROBOT NAVIGATION IN
CHALLENGING REAL-WORLD
INDOOR AND OUTDOOR ENVIRONMENTS**

Adarsh Jagan Sathyamoorthy
Doctor of Philosophy, 2024

Dissertation Directed by: **Professor Dinesh Manocha**
Department of Computer Science

The use of autonomous ground robots for various indoor and outdoor applications has burgeoned over the years. In indoor settings, their applications range from waiters in hotels, helpers in hospitals, cleaners in airports and malls, transporters of goods in warehouses, surveillance robots, etc. In unstructured outdoor settings, they have been used for exploration in off-road environments, search and rescue, package delivery, etc. To successfully accomplish these tasks, robots must overcome several challenges and navigate to their goal.

In this dissertation, we present several novel algorithms for learning-based perception combined with model-based autonomous navigation in real-world indoor and outdoor environments. The presented algorithms address the problems of avoiding collisions in dense crowds (< 1 to 2 persons/sq.meter), reducing the occurrence of the freezing robot problem, navigating in a socially compliant manner without being obtrusive to humans, and avoiding transparent obstacles in indoor settings. In outdoor environments, they address challenges in estimating the traversability

of off-road terrains and vegetation, and understanding explicit social rules (e.g. crossing streets using crosswalks).

The presented algorithms are designed to operate in real-time using the limited computational capabilities on-board real wheeled and legged robots such as the Turtlebot 2, Clearpath Husky, and Boston Dynamics Spot. Furthermore, the algorithms have been evaluated in real-world environments with dense crowds, transparent obstacles, off-road terrains, and vegetation such as tall grass, bushes, trees, etc. They have demonstrated significant improvements in terms of several metrics such as increasing success rates by at least 50% (robot avoids collisions and reaches its goal), lowering freezing rates by at least 80% (robot does not halt/oscillate indefinitely), increasing pedestrian friendliness up to 100% higher, reducing vibrations experienced in off-road terrains by up to 22%, etc over the state-of-the-art algorithms in various test scenarios.

The first part of this dissertation deals with socially-compliant navigation approaches for crowded indoor environments. The initial methods focus on collision avoidance, handling the freezing robot problem in crowds of varying densities by tracking individual pedestrians, and modeling regions the robot must avoid based on their future positions. Subsequent works expand on these models by considering pedestrian group behaviors. The next part of this dissertation focuses on outdoor navigation methods that estimate the traversability of various terrains, and complex vegetation (e.g. pliable obstacles such as tall grass) using perception inputs to navigate on safe, and stable terrains. The final part of the dissertation elaborates on methods designed for detecting and navigating complex obstacles in indoor and outdoor environments. It also explores a technique leveraging recent advancements in large vision language models for navigation in both settings. All proposed methods have been implemented and evaluated on real wheeled and legged robots.

AUTONOMOUS ROBOT NAVIGATION IN CHALLENGING
REAL-WORLD INDOOR AND OUTDOOR ENVIRONMENTS

by

Adarsh Jagan Sathyamoorthy

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:

Dr. Dinesh Manocha, Chair/Advisor
Dr. Huan Xu, Dean's Representative
Dr. Kaiqing Zhang
Dr. Pratap Tokekar
Dr. Michael Otte

© Copyright by
Adarsh Jagan Sathyamoorthy
2024

Dedication

To Amma, Appa, and Paati.

Acknowledgments

I owe my gratitude to a lot of people who have helped me throughout my Ph.D. First and foremost, I would like to thank my advisor, Dr. Dinesh Manocha for making me a part of GAMMA, motivating and trusting me with challenging research problems, and introducing me to so many wonderful and talented researchers. Dr. Manocha’s direction, and hard work helped me push myself beyond my self-imposed limitations. I am also extremely grateful for all the times when he took care of me, and helped me navigate through life’s lows. Additionally, I would like to thank my defense committee members Dr. Kaiqing Zhang, Dr. Pratap Tokekar, Dr. Mumu Xu, and Dr. Michael Otte for taking the time to serve as committee members. I am grateful that I got a chance to interact with all of them at some point during my Ph.D.

I would next like to thank all my collaborators, who have really been more like my best friends rather than co-authors. Specifically, Senthil Hariharan, Utsav Patel, Kasun Weerakoon, Jing Liang, Tianrui Guan, Mohamed Elnoor, and Nithish Kumar. I would like to add extra thanks to Kasun, and Mohamed for their help with hardware experiments outdoors. They have braved extremely high and low temperatures to get things done for ambitious deadlines. Hardware experiments using the Husky and Spot robots would not have been possible if not for Dr. Ivan Pinsky’s efforts at the RAL lab. I am also grateful to Jason Pusey, Mason Russell, and Dr. Damon Conover for their support, and the time and effort they put in at Graces Quarters for our yearly ArtiAMAS demos.

I would also like to thank other GAMMA members like Trisha Mittal, Rohan Chandra, Pooja Guhan, Vishnu Shashank Dorbala, and Divya Kothandaraman for their support and suggestions to improve my research. Thanks also to my flat-mate Mohammed Khalid for the indoor cricket, badminton, research and philosophical discussions.

Next, I must acknowledge the people at UMD's Electrical and Computer Engineering department. I'm thankful to Emily Irwin, Maria Hoo, and Souad Nejjar for their help after my proposal exam, which quickly advanced me to Ph.D. candidacy.

I am grateful to the funding sources (Intel, ARO Grants W911NF1910069, W911NF1910315, W911NF2110026, NSF Grant 2031901, U.S. Army Grant No. W911NF2120076) that have enabled me and my collaborators to fully dive into research. Extra thanks to the Ann Wylie Dissertation Fellowship.

Finally, I would like to thank my Amma, Appa, and Paati. Their unconditional love, trust, sacrifice, and support for anything I've wanted to do have helped me reach places that seemed light-years away. And lastly, I am grateful to the divine hand that has nudged me on this path.

Table of Contents

Preface	ii
Foreword	ii
Dedication	ii
Acknowledgements	iii
Table of Contents	v
List of Tables	viii
List of Figures	ix
List of Abbreviations	xi
Chapter 1: Introduction	1
1.1 The Navigation Task	1
1.1.1 Robot Sensing	2
1.1.2 Robot Motion Planning	4
1.2 Challenges and Opportunities	6
1.3 Main Contributions	9
1.4 Outline of the Dissertation	12
Chapter 2: Navigating Crowded Indoor Environments	13
2.1 Overview	13
2.2 Related Work	17
2.2.1 Pedestrian Tracking and Motion Models	17
2.2.2 Pedestrian Trajectory Prediction and Navigation	18
2.2.3 Learning-based Crowd Collision Avoidance	19
2.2.4 Methods Addressing FRP	20
2.2.5 Social-Compliant Navigation	20
2.3 DenseCAvoid	21
2.3.1 Preliminaries	21
2.3.2 Methodology	23
2.3.3 Results and Evaluations	29
2.3.4 Limitations	34

2.4	Frozone	34
2.4.1	Preliminaries	34
2.4.2	Methodology	37
2.4.3	Results and Evaluations	46
2.4.4	Limitations	52
2.5	Summary	53
Chapter 3:	Navigating Amongst Groups in Indoor Environments	55
3.1	Overview	55
3.2	Related Work	60
3.2.1	Social Distancing Detection	60
3.2.2	Group Detection	61
3.2.3	Group Behaviors and Interaction Detection	61
3.3	COVID Surveillance Robot	62
3.3.1	Detecting Social Distancing Breaches	62
3.3.2	Results and Evaluations	73
3.3.3	Limitations	86
3.4	CoMet	86
3.4.1	Notations and Definitions	88
3.4.2	Modeling Group Cohesion	89
3.4.3	Cohesion-based Navigation	97
3.4.4	Results and Evaluations	100
3.4.5	Limitations	106
3.5	Summary	107
Chapter 4:	Navigating in Unstructured Outdoor Environments	108
4.1	Overview	108
4.2	Related Work	114
4.2.1	Characterizing Terrain Traversability	114
4.2.2	Navigating Unstructured Terrains	116
4.2.3	Perception in Dense Vegetation	117
4.2.4	Navigating Dense Vegetation	118
4.3	TerraPN	119
4.3.1	Preliminaries	119
4.3.2	TerraPN: Predicting Surface Costs	121
4.3.3	TerraPN: Navigation	127
4.3.4	Results and Analysis	131
4.3.5	Limitations	139
4.4	VERN	140
4.4.1	Preliminaries	140
4.4.2	VERN: Vegetation Classification	143
4.4.3	VERN: Navigation in Dense Vegetation	146
4.4.4	Results and Analysis	151
4.4.5	Limitations	157
4.5	Summary	158

Chapter 5:	Perceiving Complex Obstacles and Contexts in Indoor and Outdoor Environments	160
5.1	Overview	160
5.2	Related Work	164
5.2.1	Sensing using Point Cloud Intensity	165
5.2.2	Detecting Indoor Obstacles	166
5.2.3	Detecting Outdoor Obstacles	167
5.2.4	Vision Language Models in Navigation	167
5.3	Multi-layer Intensity Maps	169
5.3.1	Background	169
5.3.2	MIM: Approach	171
5.3.3	Results and Evaluations	178
5.3.4	Limitations	183
5.4	CoNVOI	184
5.4.1	Background	184
5.4.2	CoNVOI: Context-aware Navigation using VLMs in Outdoors and Indoors	186
5.4.3	Results and Evaluations	190
5.4.4	Limitations	199
5.5	Summary	199
Chapter 6:	Conclusion	201
	Bibliography	204

List of Tables

2.1	Ablation Experiments of using Pedestrian Trajectory Prediction	30
2.2	Comparing Methods for Handling Freezing Scenarios	30
2.3	Frozone’s Navigation Performance Compared with Prior Methods	48
2.4	Comparing Robot’s Freezing Rates and Pedestrian Friendliness	49
3.1	Comparison of Breach Detection and Enforcement for Various Configurations . .	80
3.2	CS-Robot’s Social Distancing Breach Detection Accuracy	81
3.3	CS-Robot’s Pedestrian Tracking Duration with Varying Pedestrian Velocities . . .	84
3.4	List of Symbols Used in CoMet and their Definitions.	89
3.5	Results of CoMet’s Group Classification Based on their Cohesion	104
3.6	Comparisons of Navigation Metrics for Various Methods and CoMet-based Navigation	105
4.1	Quantitative Comparisons of Navigation Performance of Various Methods with TerraPN	135
4.2	TerraPN’s Inference and Training Times Compared with Various Methods	135
4.3	Quantitative Navigation Comparisons between VERN and Other Methods	155
4.4	VERN’s Vegetation Classification Accuracy	156
5.1	Quantitative Comparison between using MIMs Versus Other Methods for Navigation	180
5.2	Comparison of Inference and Training Time between MIMs and Other Methods .	181
5.3	Ablation Studies Demonstrating the Benefits of Context-based Prompting and Multi-modal Visual Marking	195
5.4	Quantitative Navigation Comparisons of CoNVOI with Prior Methods	196

List of Figures

2.1	DenseCAvoid Operating in Dense Crowds	15
2.2	Instance of Freezing and Frozone’s Deviation Velocity	16
2.3	Pedestrian Tracking and Future Position Prediction	22
2.4	DenseCAvoid’s Neural Network Architecture	26
2.5	DenseCAvoid’s Simulated Training Scenarios	27
2.6	Convergence of DenseCAvoid’s Reward Function	29
2.7	Formulation of the Freezing Robot Problem	38
2.8	Frozone’s Deviation Angle	45
2.9	Frozone’s Overall Architecture	47
2.10	Frozone’s Freezing Avoiding Trajectories	52
3.1	CS-robot Detecting Social Distancing Breaches.	58
3.2	Detecting the Cohesion of Groups for Robot Navigation	59
3.3	CS-Robot’s Overall Architecture	64
3.4	Criteria for a Social Distancing Breach	65
3.5	Pedestrian Localization Using RGB-D Camera	66
3.6	Homography and Coordinate Frames in CS-Robot	67
3.7	Thermal Camera Images Used to Detect High Body Temperatures	73
3.8	CS-Robot’s Pedestrian Localization Accuracy	78
3.9	CS-Robot’s Breach Detection Computation Time	83
3.10	Improved Pedestrian Tracking Using CCTV Camera	85
3.11	Features Used to Compute the Group Cohesion Metric	90
3.12	CoMet-based Socially-Compliant Navigation Architecture	92
3.13	CoMet-based Navigation’s Deviation Velocity Away from Group PFZ	95
3.14	Qualitative Comparisons of Trajectories of Various Methods with CoMet-based Navigation	101
3.15	CoMet-based Navigation in Real-world Scenarios	102
4.1	TerraPN Navigating Unstructured Terrains with Various Surface Properties	112
4.2	VERN Navigating Spot through Dense Tall Grass	113
4.3	TerraPN’s Overall Architecture	119
4.4	Principal Component Analysis on 6-DOF IMU Data	123
4.5	TerraPN’s Neural Network Architecture	124

4.6	TerraPN’s Surface Cost Map Predictions Compared with Semantic Segmentation Methods	133
4.7	Qualitative Comparisons of Trajectories between TerraPN and Various Methods	136
4.8	VERN’s Overall Architecture	143
4.9	VERN’s Siamese Neural Network Architecture	145
4.10	VERN’s Vegetation Classification Results and Vegetation-aware Cost Maps	151
4.11	Qualitative Comparisons of Trajectories between VERN and Other Methods in Dense Vegetation	152
5.1	MIM Used to Navigate Spot through Dense Vegetation	163
5.2	CoNVOI Using Crosswalks to Cross the Street and Following Detour Signs	165
5.3	MIM’s Overall System Architecture	172
5.4	Uniform Versus Adaptive Inflation of Obstacles	175
5.5	MIM Detecting and Appropriately Handling Glass, String Curtains, Tall Grass	177
5.6	CoNVOI’s Overall Architecture	185
5.7	CoNVOI’s Context-aware Trajectories Compared with Prior Methods	191
5.8	Reference Trajectories Generated by GPT-4v and Gemini	192

List of Commonly Used Abbreviations

CNN	Convolutional Neural Network
CoMet	Cohesion Metric
CS-Robot	COVID Surveillance Robot
DL	Deep Learning
DRL	Deep Reinforcement Learning
DWA	Dynamic Window Approach
FOV	Field of View
FRP	Freezing Robot Problem
GPT	Generative Pre-training Transformer
LiDAR	Light Detection and Ranging
LLM	Large Language Model
MIM	Multi-layer Intensity Map
ORCA	Optimal Reciprocal Collision Avoidance
PFZ	Potential Freezing Zone
POMDP	Partially Observable Markov Decision Process
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
RVO	Reciprocal Velocity Obstacles
VO	Velocity Obstacles
VLM	Vision Language Model
YOLO	You Only Look Once

Chapter 1: Introduction

1.1 The Navigation Task

Mobile robots range from simple transport vehicles that carry heavy pallets in large warehouses, small boxes on wheels that deliver food in office and college campuses, to lunar and Mars rovers that survive extreme extra-terrestrial environments. The last several decades have seen the explosive development of such robots and their applications in warehouse and inventory management [1], delivery robots [2] and cleaners [3, 4] in hospitals, hotels, airports, malls, subways, surveillance [5], agriculture [6], planetary exploration [7], search and rescue [8], to name a few.

An important reason for this widespread application is their versatility. Mobile robots can have different abilities to carry loads, operate for different durations, and traverse through different types of indoor and outdoor environments. These capabilities depend on factors such as the size of the robot, its dynamics, and whether it is wheeled, tracked, or legged. To leverage their hardware mobility for present and future applications, mobile robots must possess the capability to be autonomous (perform tasks without human assistance).

Making a robot autonomous necessitates the integration of many specialized components depending on the application. However, broadly, a robot's autonomy is achieved through a set of foundational components. They include tightly-knit components for *sensing* various entities

in the environment, *path planning* that computes a set of waypoints connecting the robot's start location to a goal location (e.g. such as Google Maps that provides a path), *motion planning* that computes actions (e.g. robot velocities) the robot can take to follow waypoints (e.g. like a driver who follows the directions from Google Maps), *mapping and localization* that create a complete map of the environment and estimate the robot's pose relative to the map as it moves, and low-level *control* that actually executes the actions from the motion planner on the robot. Of these components, the robot's sensing and motion planning typically deal with local or near-vicinity challenges such as detecting and avoiding dynamic obstacles around it. This dissertation presents significant improvements to the local sensing and motion planning components of robots. Motion planning is broadly referred to as *navigation*.

1.1.1 Robot Sensing

Mobile robots perceive the world using a variety of visual and non-visual sensors. For example, visual sensors such as Red-Green-Blue (RGB) and Depth (RGB-D) cameras that provide color and depth images, respectively, have been used extensively for computer vision research. Recent research that use such images include detecting different kinds of objects/obstacles [9], detecting and tracking humans [10–12], their body poses [13, 14], emotions [15], detecting transparent and reflective objects [16, 17], identifying terrains [18], visual landmarks [19, 20], etc. RGB images have been employed in all the aforementioned applications because they provide rich information about the environment such as color, texture, object shape and size, etc akin to the human eye. However, RGB images struggle to record information when the environmental lighting is low, leading to perception inaccuracies. Depth images provide an additional dimension

of information with the distances of objects from the camera at each pixel location by either using the time-of-flight [21] of lasers projected from the camera, or using a pair of RGB cameras for stereo vision [22]. This is especially helpful for object/obstacle localization in the environment.

Another class of sensors gathers environmental information by shooting radio waves (Radio Detection and Ranging (RADAR)), sound waves (Sound Navigation and Ranging (SONAR)), or laser rays (Light Detection and Ranging (LiDAR)), and detecting their reflections from surrounding objects. This way, the objects' proximity and in some instances their shapes can be discerned. Particularly, LiDARs, which can either be 2D or 3D based on whether the laser is shot on a 2D plane or a set of planes in 3D space, have been used extensively for detecting a robot's proximity to obstacles. The sensing modality from 2D and 3D LiDARs are referred to as laser scans and point clouds, respectively. These modalities contain lists of points with the (x, y) or (x, y, z) coordinates, corresponding to objects in the LiDAR's vicinity. Additionally, LiDARs possess high fields-of-view (FOV) ($180^\circ - 360^\circ$) that help detect objects around the robot.

The raw laser scans and point clouds from LiDARs have been transformed into occupancy grid maps, or cost map representations [23–25] of the robot's vicinity. These maps consist of a set of grids that each represent a discretized location near the robot, and contain the probability of an obstacle being present at that location. Additionally, the values in such grid maps can be customized to represent attributes such as uncertainties in obstacle detection [26], terrain elevation [27], terrain's surface roughness, bumpiness [28], etc. Oftentimes, the values in such map representations are directly used to calculate the *cost* of a robot traversing a region.

While visual modalities aid in detecting the states of objects around the robot, several non-visual modalities play a crucial role in detecting the robot's own state. Such sensors include Inertial Measurement Units (IMUs), odometry sensors such as wheel or joint encoders, etc. IMUs

consist of accelerometers, gyroscopes, and magnetometers that help measure the pose (roll, pitch, and yaw angles) of the robot, and the vibrations it experiences. Wheel and joint encoders gauge the rotations or translations of joints to estimate the distance the robot has moved from its initial position.

The research detailed in this dissertation utilizes the aforementioned sensor modalities in varying capacities to address different perception and navigation challenges.

1.1.2 Robot Motion Planning

Using the states of the robot and the entities in its surroundings, a motion planner computes actions/velocities to primarily 1. avoid obstacles, 2. reach the robot’s goal/waypoint, 3. be locally optimal in terms of an objective. For example, the objective can be time to reach the goal, reaching the goal in the shortest distance, smoothness of the robot’s motion, etc. Numerous model-based [29–33] and learning-based [34–39] planning algorithms have been proposed with these capabilities. The actions/velocities outputted by a motion planner is generally a velocity pair (v, ω) for non-holonomic, differential drive systems or (v_x, v_y) for holonomic systems, where v and ω denote a robot’s linear and angular velocities, respectively. v_x and v_y denote the linear velocities in the X (forward) and Y (leftward) directions relative to the robot.

Model-based planning approaches explicitly formulate a model that accounts for the robot’s and obstacles’ positions, velocities, and accelerations and compute robot velocities that locally maximize or minimize a hand-crafted objective function. A few popular examples are Velocity Obstacle (VO) [40] such as Reciprocal VO [31] and ORCA [33] that provide decentralized collision avoidance by locally altering the trajectories for agents with single-integrator dynamics.

The Dynamic Window Approach (DWA) [29] is another method that formulates a search space of velocities called dynamic window based on the obstacle locations, and the robot’s acceleration constraints, and searches for a (v, ω) that minimizes a cost function with costs for goal-reaching, obstacle-avoiding, forward-moving. Potential Fields-based methods [32,41] formulate the motion planning task similar to electrical/magnetic fields, where the robot experiences attractive “forces” towards its goal, and repulsive “forces” away from obstacles. Model-based approaches can guarantee that their velocities will avoid collisions and lead to other behaviors, and are explainable. That is, one can precisely interpret why a certain action was performed given the robot and environmental states.

Such navigation capabilities have also been achieved by approaches that use various kinds of Deep Learning (DL) techniques to train Deep Neural Networks (DNNs) to implicitly model the relationships between the robot’s state, the obstacles’ states, and the goal location. For instance, [34] trained a scalable, sensor-level collision avoidance policy using Deep Reinforcement Learning (DRL) that uses raw laser scans as inputs to detect obstacles, and output a (v, ω) pair. It was extended to a hybrid learning architecture [35], which switched policies based on the obstacle density in the environment. Cooperative behaviors between humans and robots have also been modeled using DRL and a value network for improved collision avoidance [36]. Other kinds of learning such as Imitation Learning [42] have also been used to train navigation policies. Learning-based approaches have shown to be resilient to uncertainties in obstacle state estimation [34,35,38], and possess superior capabilities to model complex relationships [43,44] between the motion of special obstacles such as humans, and the robot. However, training these approaches stably, and quickly is complicated [45]. DRL-based approaches especially are data inefficient [46] during training. Therefore, they are trained iteratively in simulated environments. However,

such methods and learning-based navigation in general suffer from the sim-to-real gap [47–49], where a model or policy’s performance in simulation deteriorates in the real world. Additionally, the actions of such methods are not easily explainable, and do not guarantee that capabilities such as collision avoidance and goal-reaching would be exhibited under all circumstances.

1.2 Challenges and Opportunities

Given the background of sensing and motion planning, this section discusses the major challenges that real-world indoor and outdoor environments pose to robot navigation. Navigating in static or slightly dynamic scenarios with smooth, flat ground has been extensively studied over the years [29, 50–52]. However, such methods cannot reliably handle dense, and dynamic scenarios such as indoor settings with pedestrian crowds, or account for complex terrain properties such as elevation changes, surface roughness, bumpiness, deformability, etc in outdoor settings. Moreover, the challenges in sensing and navigation in indoor and unstructured outdoor settings vary significantly.

Indoor settings typically present static obstacles such as walls, furniture, narrow corridors with T and L junctions, and dynamic obstacles such as humans. Due to the presence of a multitude of obstacles, a robot’s navigation faces the following challenges:

- **Dense Scenarios:** The presence of static and dynamic obstacles in close proximity significantly reduces the amount of free space available to the robot for navigation. Such scenarios are considered *dense*. This reduces the number of candidate velocities/actions that a robot can perform to avoid collisions and reach the goal.
- **Freezing Robot Problem (FRP):** An important phenomenon that occurs in dense scenarios

where a robot's motion planner deems all possible velocities lead to collisions. This completely halts the robot for an extended period of time or causes oscillatory behaviors, and is resolved only when the nearby obstacles (maybe humans) move to give way.

- **Social Compliant Behaviors:** Humans navigate by following several implicit social rules that consider other humans' need for space, avoid cutting in front of other walking humans, etc. Therefore, when a robot navigates in the presence of humans, they must be considered differently compared to inanimate obstacles. Behaviors such as moving too close to pedestrians during collision avoidance, disrupting their motion, cutting through a group of people, etc must be avoided.
- **Detecting Transparent Obstacles:** Modern buildings include a lot of transparent and reflective surfaces and objects such as glass walls, large mirrors, glass sculptures, etc. They can be detected with sound-based ranging sensors such as ultrasonic sensors. However, such sensors typically have a short range and low FOV that may be insufficient for navigation. Furthermore, commonly used RGB/RGB-D images, laser scans, and point clouds cannot detect transparent and reflective obstacles as most light either passes through or reflects away from them and are not captured by cameras, LiDARs, etc. Yet, it would be beneficial for robot navigation if these visual sensors could somehow be used to detect transparent obstacles.

Unstructured outdoor settings present a different set of navigational challenges mainly based on an off-road terrain's geometric and surface properties, and the other kinds of obstacles not found in indoor environments such as vegetation. The associated challenges are:

- **Terrain Elevation:** A terrain's slope and elevation changes pose the risks of unsafe pitch

and roll orientations for a ground robot, and even cause flip-overs. Therefore, the elevations at various regions must be accounted for while traversing them.

- **Terrain Surface Properties:** Apart from a terrain's slope and elevation, its surface properties (texture, bumpiness, and deformability) govern its navigability for a robot. For instance, a surface's texture determines the traction experienced by the robot, its bumpiness determines the vibrations experienced, and deformability determines whether a wheeled robot could get stuck or experience wheel slips (e.g. in mud). Therefore, such properties must be estimated in advance and the most navigable terrains must be chosen for traversal.
- **Vegetation:** Most unstructured outdoor environments have different kinds of vegetation such as short and tall grass, bushes, trees of various shapes, etc. In applications such as exploration, dense vegetation may have to be traversed, and the robot may not find enough free space to circumvent the flora. In such cases, the robot must estimate if certain vegetation are traversable. For instance, tall grass bends when humans or vehicles move through it and contact with such soft, pliable objects do not damage the robot, whereas bushes and trees are solid, *non-pliable* and should be avoided.
- **Interpreting Contextual Cues:** In structured and paved environments (e.g. urban, sub-urban settings), apart from picking the most navigable terrain, the robot may have to identify and interpret certain explicit and implicit social cues. These include identifying crosswalks to cross a street, or detecting sign boards (e.g. detours), etc and adapting the navigation behaviors accordingly.

While the challenges mentioned above stem from environmental factors, additional challenges emerge from limitations in sensing and computation on board the robot. Edge computers mounted

on robots are restricted to have low power consumption, which also affects their processing speeds. Additionally, sensors such as RGB/RGB-D cameras are typically limited by their low FOV ($\leq 120^\circ$). Developing algorithms that function in real-time, with limited sensing and low computational demands, and can operate in real-world environments presents another significant practical challenge.

1.3 Main Contributions

The challenges detailed above encompass a large number of sub-problems and corner cases that arise in the real world. The scope of this dissertation is limited to exploring and formulating closely knit sensing and navigation methods that address a broad subset of these problems. The main contributions of this dissertation are:

- *Methods for addressing FRP and indoor social-compliant navigation:* Two methods [53, 54] focusing on dealing with collision avoidance and FRP were developed considering the pedestrians as *individual* obstacles. Subsequently, by extending the method in [54], an approach to check for social distancing in *groups* of people using a mobile robot was proposed during the COVID-19 pandemic. Furthermore, [55], a novel method to understand pedestrian group dynamics, and navigate in highly dense environments was proposed. In detail, the proposed methods are:

1. We present DenseCAvoid, an end-to-end DRL navigation model that uses visual sensors and a pedestrian trajectory prediction algorithm to anticipate their motion, avoid collisions, and reduce the occurrence of FRP.

2. We present Frozone, a model-based approach that predicts pedestrians' trajectories, constructs a *Potential Freezing Zone (PFZ)* where the robot has a high probability of freezing and being obtrusive to their motion, and computes a deviation angle to avoid it.
 3. We present CS-Robot, a method to detect pairs of humans in a crowded scenario who are not maintaining social distancing, i.e. about 2 meters of space between them using an autonomous mobile robot and existing CCTV (Closed-Circuit TeleVision) cameras. CS-Robot uses Frozone to navigate to non-compliant groups of people to encourage them to move apart.
 4. We present CoMet, an approach to estimate a pedestrian group's cohesion (a measure of interpersonal *closeness*) to improve a robot's navigation by accounting for different levels of group cohesion while a robot moves through a crowd.
- *Methods for estimating unstructured terrain and vegetation traversability:* Two methods, one focusing on unstructured terrain traversability estimation for a wheeled mobile robot, and the other focused on legged robot navigation in densely vegetated environments. In detail, the proposed methods are:
 1. We present TerraPN, a novel method that learns the surface properties (traction, bumpiness, deformability, etc.) of complex outdoor terrains directly from robot-terrain interactions through self-supervised learning, and uses it for autonomous robot navigation. TerraPN uses RGB images of terrain surfaces and the robot's velocities as inputs, and the IMU vibrations and odometry errors experienced by the robot as labels for self-supervision.

2. We present VERN, a novel method for autonomous legged robot navigation in densely vegetated environments with a variety of pliable/traversable and non-pliable/untraversable vegetation. We present a novel few-shot learning classifier that can be trained on a few hundred RGB images to differentiate flora that can be navigated through (e.g. tall grass), from the ones that must be circumvented (e.g. bushes, trees, etc).
- *Methods that can sense and navigate in both indoor and outdoor environments:* While previously proposed methods were focused on either the indoor or outdoor settings specifically, the final proposed methods focus on sensing and navigation in both settings.
1. We present MIMs (Multi-Layer Intensity Map), a 3D object representation for robot perception and autonomous navigation. MIMs consist of multiple stacked layers of 2D grid maps each derived from reflected point cloud intensities corresponding to a certain height interval. The different layers of intensity maps can be used to simultaneously estimate obstacles' height, solidity/density, and opacity in indoor and outdoor environments.
 2. We present CoNVOI, a novel method for autonomous robot navigation in real-world indoor and outdoor environments using Vision Language Models (VLMs). A compact VLM is used to identify the *context* or scenario of the robot's environment, a multi-modal visual marking method to annotate RGB images with markers, and finally a state-of-the-art large VLM is to compute a suitable trajectory given the identified context.

1.4 Outline of the Dissertation

The dissertation is organized as follows; Chapter 2 discusses the contributions related to collision avoidance, and navigating in a social-compliant manner in crowded indoor settings, and addressing key challenges such as FRP. Chapter 3 presents methods that detects groups in crowds and their behaviors to improve a robot’s social-compliant navigation capabilities. Chapter 4 describes the contributions made towards using wheeled and legged robots to address two distinct challenges in unstructured outdoor scenarios: terrain properties and dense vegetation. Chapter 5 first explains contributions made towards sensing complex obstacles such as transparent objects, thin, pliable obstacles such as string curtains in indoors and tall grass in outdoors. Next, it details a method that enables new navigation behaviors such as crossing streets at crosswalks, moving on paved surfaces such as sidewalks, not interrupting human groups, etc to be realized easily without any reformulation. Finally, Chapter 6 discusses the limitations of the proposed work, and directions for future work.

Chapter 2: Navigating Crowded Indoor Environments

2.1 Overview

Mobile robots are increasingly used in many indoor scenarios. This includes waiting and delivering food in restaurants and hotels, remotely monitoring patients in hospitals, transporting goods in warehouses, surveilling, delivering packages, etc. To accomplish such tasks, these robots may have to navigate through dense and challenging dynamic environments, specifically in crowds with pedestrian densities ranging from < 1 to 2 persons/m².

The problem of robot navigation among dynamic obstacles has been well studied in robotics and related areas. There is a large body of work on *model-based navigation techniques* based on potential fields, velocity obstacles, and dynamic windows [31, 33, 50, 56, 57]. Recently, many collision avoidance methods based on machine learning have also been proposed [34, 36, 42, 43, 58, 59] and have shown considerable promise in real-world scenarios. These *learning-based methods* can be directly integrated with existing 2-D or 3-D LiDARs or cameras and are robust to sensing inaccuracies in the states of the obstacles.

In practice, dense crowds pose several challenges for robot collision avoidance. First, pedestrian motions in such crowds can be highly non-smooth [60]. Second, the robot must be able to react to sudden changes in pedestrian motion to avoid collisions. Current learning-based collision avoidance methods [36, 42, 61] work well for sparse or moderately dense crowds, but

either result in collisions or oscillations [34] as the crowd density increases.

Apart from avoiding collisions, the robot should also navigate in a pedestrian-friendly manner. The latter includes satisfying certain social constraints [62–64], such as maintaining sufficient distance from pedestrians and other rules corresponding to avoiding them from behind (see Fig. 2.1 Bottom).

Another challenging problem that the robot must overcome in such scenarios is the Freezing Robot Problem (FRP) [65, 66]. FRP occurs when the robot faces a situation where the collision avoidance module declares that all possible velocities may lead to collisions. The robot either halts or starts oscillating indefinitely, which could either result in a collision or it does not make progress towards its goal. In practice, it is non-trivial to completely avoid FRP in crowds beyond a certain density without human cooperation [66]. One of the goals is to develop approaches that can reduce the occurrence of FRP in moderately dense crowds (≤ 1 person/ m^2) without assuming human cooperation, and using limited sensing capabilities.

There have been a few works addressing FRP [65–67]. Some approaches have also attempted to solve the problems of freezing and loss of localization in a crowd simultaneously [68]. While these methods are promising, we need more general solutions that rely less on global information and can also provide some guarantees on the resulting performance.

We contribute the following methods to address these crucial problems:

1. DenseCAvoid (section 2.3, Fig. 2.1), an end-to-end DRL-based crowd navigation policy that utilizes a pedestrian trajectory prediction algorithm to anticipate their motions and generate smooth robot trajectories to avoid them. DenseCAvoid employs a novel neural network structure and reward function that uses multiple sensor inputs and pedestrian



Figure 2.1: **[Top]** DenseCAvoid implemented on a Turtlebot navigating through a dense crowd. **[Bottom]** We explicitly track and predict pedestrian motions (marked for three pedestrians) and use it to train a Deep Reinforcement Learning-based collision avoidance policy. Our network implicitly learns to reduce the occurrence of the freezing robot problem and generates smooth robot trajectories in dense crowds.

prediction data to train the policy. Further, it reduces the occurrence of FRP by 48% in complex 3-D simulations of indoor environments with pedestrians and static obstacles.

2. Frozone, a real-time algorithm that significantly reduces the occurrence of FRP by explicitly predicting pedestrian trajectories, classifying them as *potentially-freezing* or *non-freezing* pedestrians and constructing a Potential Freezing Zone (PFZ). Our method ensures that the robot’s velocity avoids the PFZ, and is unobtrusive to the nearby pedestrians based on their social and psychological constraints for personal space. This leads to significantly more pedestrian *friendly* trajectories (section 2.4.3.3). Frozone is combined with a DRL-based

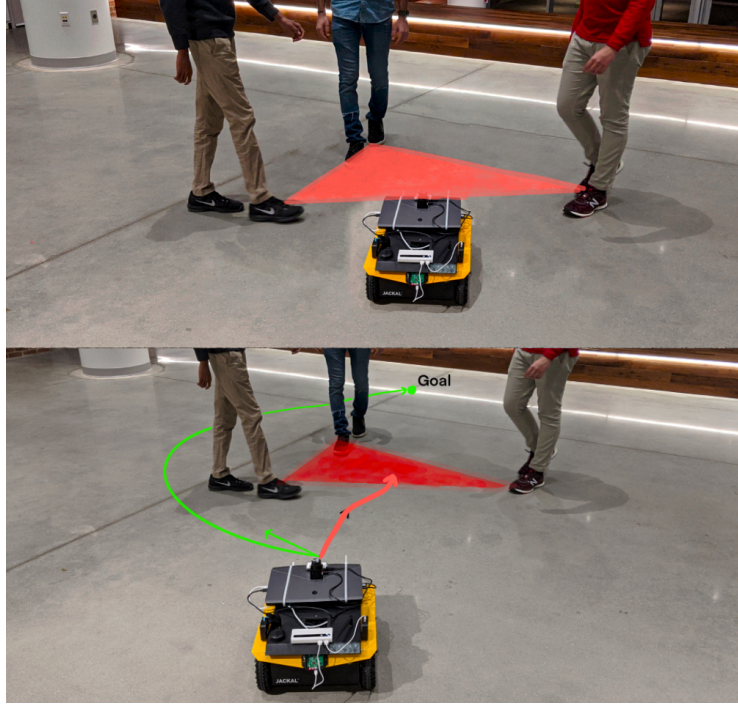


Figure 2.2: **[Top]** An instance of the Freezing Robot Problem (FRP), where the robot halts or oscillates indefinitely when it faces scenarios with pedestrians, as shown (highlighted in red). **[Bottom]** Another scenario where Frozone implemented on a Clearpath Jackal robot navigates amongst pedestrians and preemptively avoids FRP. Frozone explicitly tracks and predicts pedestrians’ positions in the sensing range of the robot and classifies each pedestrian as *potentially-freezing* or *non-freezing*. We compute a *potential* freezing zone (PFZ) (shown in red) and our method deviates the robot away from it (green trajectory), while previous methods move the robot towards that zone (red trajectory).

collision avoidance method and presents a hybrid navigation algorithm that combines the benefits of traditional model-based algorithms such as better guarantees, and DRL-based approaches like better robustness to sensor uncertainty.

Both methods have also been implemented on real robots such as the Turtlebot 2 and Clearpath Jackal, and evaluated in crowded indoor environments qualitatively.

The rest of the chapter is structured as follows. We discuss prior work relevant to navigating in dense crowds, predicting pedestrian trajectories, social navigation, etc in section 2.2. Then in sections 2.3 and 2.4, we discuss DenseCAvoid’s, and Frozone’s methodologies and evaluations

in detail, respectively. We conclude this chapter with a summary of the contributions and their limitations.

2.2 Related Work

In this section, we cover the prior work in pedestrian tracking, motion prediction, and learning-based collision avoidance methods that form the basis for DenseCAvoid [53]. Further, we detail the works that have handled freezing and social-compliant navigation methods that Frozone [54] builds upon.

2.2.1 Pedestrian Tracking and Motion Models

Object and pedestrian detection has been widely studied in computer vision. Some of the most accurate methods are based on deep learning including R-CNN [69] and its faster variants [70–72], which use a selective search area to optimize the object detection problem. Other works in learning-based tracking include [73–76]. Many of these learning-based methods lack real-time performance that is needed for navigation in dense environments. Moreover, highly accurate methods such as [12] and [77] require high-quality detection features for reliable performance.

Several motion models have been used to improve pedestrian tracking accuracy [78–81]. However, most of these methods assume a constant linear velocity or acceleration models for the pedestrians. These methods cannot characterize pedestrian dynamics accurately in dense settings [82]. Non-linear motion models such as RVO (Reciprocal Velocity Obstacles) [31] and its variants have been shown to work well for tracking in dense crowd videos. Social Force model [83], LTA [84], and ATTR [85] are other non-linear motion models that have been used

for pedestrian tracking in low to medium-density crowds.

2.2.2 Pedestrian Trajectory Prediction and Navigation

There has been extensive research in predicting object or pedestrian trajectories in computer vision and robotics. Early works include formulations such as Bayesian [86], Monte Carlo simulation [87], Hidden Markov Models [88], and Kalman Filters [89]. Deep learning-based prediction methods mostly utilize Recurrent Neural Networks (RNNs) [90] and Long Short-Term Memory (LSTM). Hybrid methods using a combination of RNNs and other deep learning architectures such as Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs) and LSTMs have also been proposed. For instance, GANs have been used for pedestrian trajectory prediction [44] and CNNs have been used for traffic prediction [91]. There has also been extensive work on accurately modeling crowd behavior [92–94] which could aid robot navigation.

Pedestrian prediction using Bayesian estimation and modeling their motions using RVO is presented in [95]. Long-term path prediction based on Bayesian learning and personality trait theory for socially-aware robot navigation is presented in [63]. Other techniques use a Partially Observable Markov Decision Process (POMDP) to model the uncertainties in the intentions of pedestrians. [96] presents a POMDP-based planner to estimate the pedestrians’ goals for autonomous driving. The planner was then augmented with an ORCA-based pedestrian motion model [97]. The resulting POMDP planner runs in near real-time.

2.2.3 Learning-based Crowd Collision Avoidance

In recent years, several works have used different learning methods for navigation in dense scenes. GAIL (Generative Adversarial Imitation Learning) [42] used raw depth camera images to train a socially acceptable navigation method for a differential drive robot. Similarly, CNNs with RGB images have been used to train an end-to-end visuomotor navigation system [58] and a deep double-Q network (D3QN) has been used to predict depth information from RGB images for static obstacle avoidance [98]. A method to use expert demonstrations in simulation to train a method for mapless navigation [99] has also been demonstrated. These methods, however, have not been evaluated in dense crowds, and may not work well in such scenarios.

A decentralized, scalable, sensor-level collision avoidance method was trained in [34]. It was extended to a hybrid learning architecture [35], which switched policies based on the obstacle density in the environment. This work was further augmented to learn localization recovery points in the environment to solve the loss of localization and freezing robot problems simultaneously [100]. Cooperative behaviors between humans and robots have been modeled using a value network for better collision avoidance [36]. This formulation was extended to observe an arbitrary number of pedestrians in the surroundings using LSTMs [61]. [43] presented an approach to model interactions within a crowd which indirectly affect robot navigation. A novel collision avoidance method that identified previously unseen scenarios to carefully navigate around pedestrians has been presented in [101].

2.2.4 Methods Addressing FRP

The earliest works addressing FRP [65, 66] argue that it can only be solved by accounting for human-robot cooperation [102]. These methods also show that even perfect pedestrian trajectory prediction would not help in solving FRP, unless human-robot cooperation is explicitly modeled. Other attempts at solving FRP include a method based on learning from demonstration [103] and improved motion prediction based on the Markov Decision Process [67]. The resulting algorithms use static CCTV cameras or simulations to validate their methods and do not fully account for many practical issues that arise on a real robot such as the loss of sensing data due to the robot’s motion.

2.2.5 Social-Compliant Navigation

Robots navigating among pedestrians should reduce the discomfort they cause to the humans in their vicinity. Prior work in crowd simulation provides insights on the psychological and environmental factors that affect the pedestrian’s trajectory [62, 104], which should be accounted during robot navigation. Works on socially aware robot navigation include predicting the long-term trajectories of pedestrians using personality trait theory and Bayesian learning [63] and classifying group emotions [105]. Learning-based methods have been used to account for social norms by specifying the behaviors that should be avoided [37]. In Frozone (section 2.4), we use these ideas and focus on computing robot velocities that avoid pedestrians from behind them, or maintain a comfortable distance in front of pedestrians.

2.3 DenseCAvoid

In this section, we explain DenseCAvoid, the end-to-end DRL navigation policy that is combined with a pedestrian motion prediction algorithm [106] to anticipate their motion and preemptively avoid them. We first detail the preliminary components such as the pedestrian tracking and Proximal Policy Optimization (PPO) [107] algorithms used in DenseCAvoid in section 2.3.1, then explain its methodology in section 2.3.2, and its evaluations 2.3.3. Finally, we discuss some of its limitations in section 2.3.4.

2.3.1 Preliminaries

2.3.1.1 Pedestrian Tracking and Trajectory Prediction

DenseCAvoid uses YOLOv3 [108], a real-time, high-accuracy variant of YOLO [109] for pedestrian detection and tracking in dense crowds. Given an RGB or a depth image and an object of interest (pedestrians, in our case), YOLOv3 outputs the bounding box coordinates over all the detected objects in the image.

To predict the future positions of the tracked pedestrians, we modify a state-of-the-art traffic trajectory prediction algorithm called RobustTP [106] to predict pedestrian motions. An image with the bounding box coordinates of the detected pedestrians (from YOLOv3) is fed as input into RobustTP. This algorithm uses a combination of CNNs and LSTMs to predict the positions of the detected pedestrians in the next frame immediately after the input image. The pedestrian tracking by YOLOv3 is shown in Fig.2.3 (a), and the position as predicted by RobustTP is shown in Fig.2.3 (b) as a black bounding box on a white background. RobustTP can also be modified to

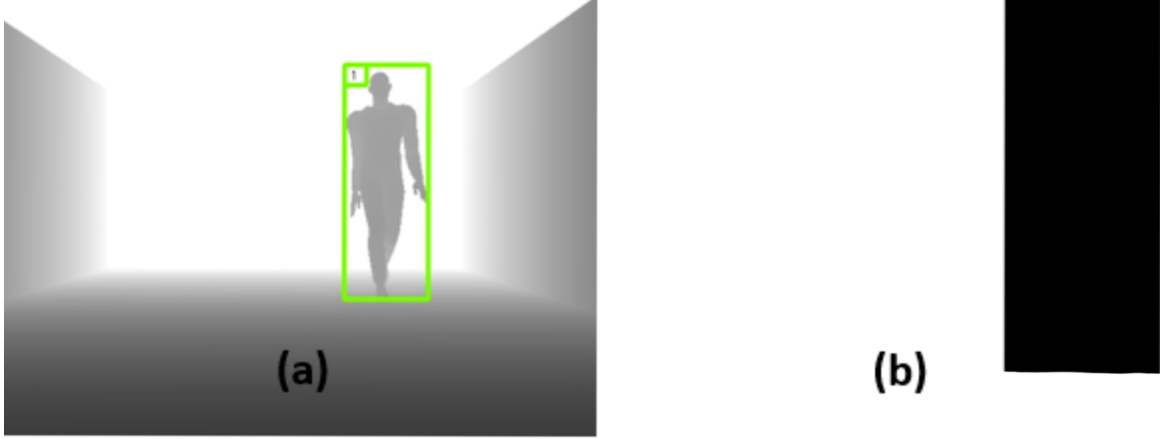


Figure 2.3: (a) Tracking a moving pedestrian in a depth image. (b) Prediction output for a future time step. The white space in the output is the admissible free space, and the black bounding box denotes the space the pedestrian would occupy in the future.

predict the trajectories of any generic obstacle. The computation time of RobustTP depends on the level of accuracy required in prediction. For mobile robot navigation in dense crowds, we set this accuracy level to $< 90\%$ to ensure real-time performance.

2.3.1.2 Deep Reinforcement Learning

DenseCAvoid is based on deep reinforcement learning, whose underlying objective is to train a policy π_θ that drives the robot to its goal while avoiding all the obstacles in the environment. During each stage of reinforcement learning training, policy parameter θ is updated based on the robot’s actions and their corresponding pre-defined rewards and penalties. There are three important components in DRL policy training, namely, **(i)** robot’s observation space, **(ii)** action space, and **(iii)** the reward function. We briefly describe our observation and action spaces here and describe our novel reward function in section 2.3.2.

Observation and Action Spaces: Our observation space at any time instant t can be represented as $\mathbf{o}^t = [\mathbf{o}_{percep}^t, \mathbf{o}_{odom}^t, \mathbf{o}_{goal}^t]$, where $\mathbf{o}_{percep}^t$ is the observation from the perception

sensors, $\mathbf{o}_{odom}^t$ is the odometry observation (which includes the current velocity of the robot), and $\mathbf{o}_{goal}^t$ denotes the goal position relative to the robot. The action space of the robot is a continuous space consisting of the linear and angular velocities of the robot, represented as $\mathbf{a}^t = [v^t, \omega^t]$.

The robot performs a certain action until it receives the observation for the next time instant $\mathbf{o}^{t+1}$. For optimizing the policy during training, we use the minimization of the mean arrival time of the robot to its goal as our objective function:

$$\operatorname{argmin}_{\pi_{\theta}} \mathbb{E} \left[\frac{1}{N} \sum_{i=1}^N t_i^g | \pi_{\theta} \right]. \quad (2.1)$$

To train our collision avoidance policy, we use a policy gradient method [110] called Proximal Policy Optimization (PPO) [107]. Policy gradient methods (in contrast with value-based methods) directly modify the policy during training, which is more suitable for navigation applications and continuous action spaces. In addition, PPO bounds the update of parameters θ to a trust region [111], thereby ensuring that the policy does not diverge between two consecutive training iterations. This guarantees stability during the training phase.

2.3.2 Methodology

In this section, we explain DenseCAvoid’s methodology that combines DRL with explicit pedestrian trajectory prediction for navigation. We present our network architecture that is used to train DenseCAvoid’s policy with anticipatory behaviors. We also discuss our reward function design and the complex 3-D training scenarios that simulate dense environments.

2.3.2.1 Anticipating Pedestrian Behavior

In densely crowded scenarios, pedestrian motion is highly non-smooth. In our case, we choose to explicitly model pedestrian behavior, similar to classic navigation methods. As mentioned in section 2.3.1.1, we use an explicit pedestrian trajectory predictor and use this information to generate non-oscillating, non-jerky navigation in dense scenarios. In addition, since our trained policy *knows* where each pedestrian is headed in the immediate future, we can also make the robot avoid regions where several pedestrians might be heading. Therefore, the robot tends to avoid scenarios that could possibly lead to the freezing robot problem.

As shown in Fig.2.3, the prediction frame extracted using [106] has black bounding boxes at locations where the pedestrians could be in the future. These boxes are placed over a white background, which represents the collision-free free space. We provide this *future* free-space representation while training our network, which makes our collision avoidance policy training converge faster. This is due to the fact that our free-space representation provides a more direct way to infer the direction the robot should move towards and to learn the dynamic properties or behaviors of the pedestrians in different settings.

In our end-to-end formulation, we integrate the pedestrian prediction outputs with the DRL collision avoidance network as a new observation. Formally, $\mathbf{o}^t = [\mathbf{o}_{lid}^t, \mathbf{o}_{cam}^t, \mathbf{o}_g^t, \mathbf{o}_v^t, \mathbf{o}_{pred}^t]$, where $\mathbf{o}_{lid}^t$ denotes raw data from a 2-D LiDAR, $\mathbf{o}_{cam}^t$ denotes the raw image data from either a depth or an RGB camera, $\mathbf{o}_g^t$ refers to the relative position of the goal with respect to the robot, $\mathbf{o}_v^t$ denotes the robot’s current velocity, and $\mathbf{o}_{pred}^t$ refers to the predicted positions of pedestrians in the next frame (shown in Fig. 2.3). Once the policy is trained, we sample a collision-free action from $\mathbf{a}^t$

at each time instant as:

$$\mathbf{a}^t \sim \pi_{\theta}(\mathbf{a}^t | \mathbf{o}^t). \quad (2.2)$$

Note that it is also possible to directly combine the prediction output with the action $\mathbf{a}^t$. This could be useful in scenarios where the robot encounters a situation that is quite different from the training data.

2.3.2.2 Network Architecture

Our network (Fig.2.4) consists of four branches, each processing a component of the observation $\mathbf{o}^t$. Two 1-D layers and three 2-D layers are used respectively for processing the 2-D LiDAR data and depth image data, which are followed by fully connected layers that modify the dimensions of the outputs of the two branches to match each other. In branch 2, the depth image from the camera is first passed into our prediction algorithm. We then stack the computed prediction frame behind a resized version of the original depth image before passing these frames through a set of three 2-D convolutional layers. ReLU activation is applied to the outputs of all the hidden layers in branches 1 and 2. Branches 3 and 4 feed the relative position of the goal and the robot’s current velocity to the fully connected layer FC2.

We apply a sigmoid activation to restrict the robot’s linear velocity between (0.0, 1.0) m/s and a tanh activation to restrict the angular velocity between (-0.4, 0.4) rad/s in the output layer. The output velocity is sampled from a Gaussian distribution that uses the mean value outputted from the fully connected layer FC2, which is updated during training. We address the imbalance between the dimensions of the perception sensors, and the robot’s current velocity and goal using appropriate training scenarios (section 2.3.2.4). Our training scenarios ensure that the velocity

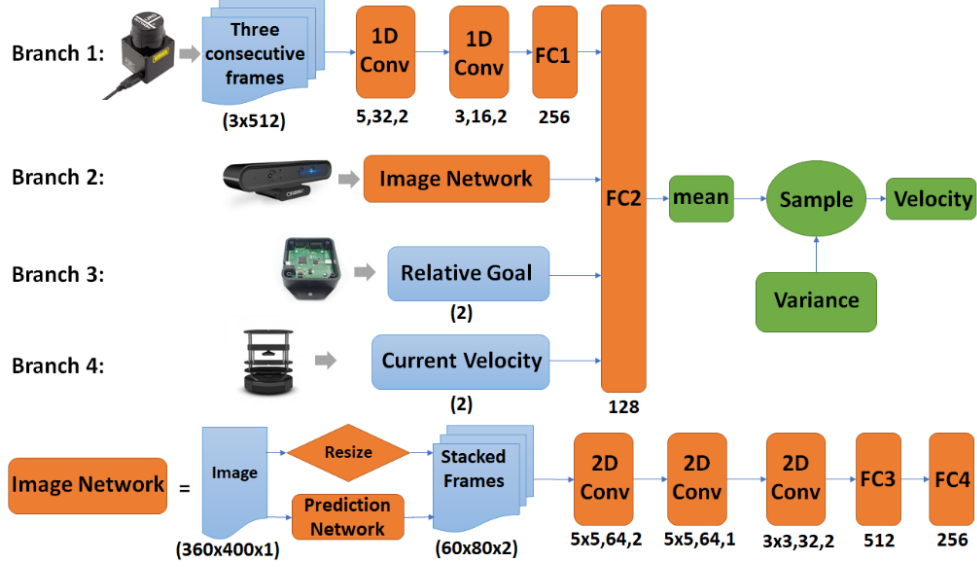


Figure 2.4: Architecture of our anticipatory collision avoidance network with four branches to process different observations. The input layer is marked in blue and, the hidden layers are marked in orange and the fully-connected layers in the network are marked as FCn. The green layer represents the output layer. The three values underneath each hidden layer denote the kernel size, number of filters, and stride length respectively. We stack a raw depth image with the prediction frame and use it directly for training our collision avoidance policy.

and goal inputs are used in the initial stages of training to learn goal reaching capabilities with reduced oscillations.

2.3.2.3 Reward Function

Our purpose during policy training is to avoid collisions while moving towards the goal and to reduce oscillations or freezing behavior during navigation. Therefore, reaching the goal and colliding with obstacles are assigned high values of reward and penalty, respectively. To obtain smooth trajectories during run-time, we penalize sudden, large changes in the angular velocity. In addition, we use intermediate waypoints to guide the policy away from obstacles and towards the goal, which results in faster convergence.

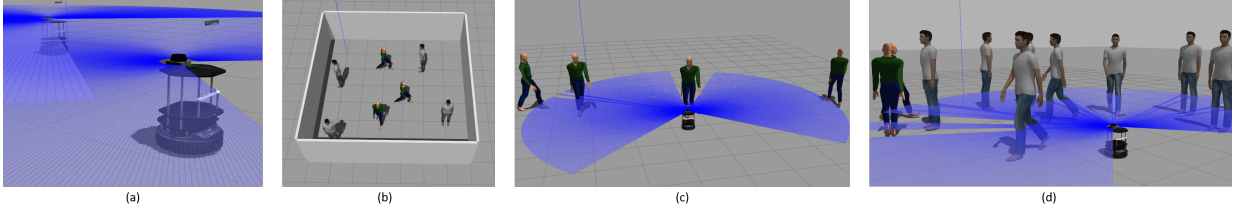


Figure 2.5: Different training scenarios used for training our algorithm from simplest to complex. **(a)**: Empty scenario with random goals; **(b)**: Dense-Static scenario with random goal; **(c)**: Robot moves through a few pedestrians walking randomly to reach the goal; **(d)** Robot moves through a dense crowd to reach its goal.

Formally, the total reward collected by a robot at time instant t can be given as:

$$r^t = (r_g)^t + (r_c)^t + (r_{osc})^t + (r_{safedist})^t, \quad (2.3)$$

where the reward for reaching the goal or an intermediate waypoint $(r_g)^t$ is given as:

$$(r_g)^t = \begin{cases} r_{wp} & \text{if } \|\mathbf{p}^t - \mathbf{p}_{wp}\| < 0.2, \\ r_{goal} & \text{if } \|\mathbf{p}^t - \mathbf{g}\| < 0.3, \\ 2.5(\|\mathbf{p}^{t-1} - \mathbf{g}\| - \|\mathbf{p}^t - \mathbf{g}\|) & \text{otherwise.} \end{cases} \quad (2.4)$$

Here, $\mathbf{p}^t$ and $\mathbf{p}_{wp}$ denotes the position of the robot at time t and the position of the waypoint respectively, and $\mathbf{g}$ denotes the robot's goal location. The collision penalty $(r_c)^t$ is given as:

$$(r_c)^t = \begin{cases} r_{collision} & \text{if } \|\mathbf{p}^t - \mathbf{p}_{obs}\| < 0.3, \\ 0 & \text{otherwise.} \end{cases} \quad (2.5)$$

The oscillatory behaviors (i.e. choosing sudden large angular velocities) are penalized as:

$$(r_{osc})^t = -0.1|\Delta\omega^t| \quad \text{if } |\Delta\omega^t| > 0.3. \quad (2.6)$$

Here, ω^t is the angular velocity of the robot. The penalty for moving too close to an obstacle is given by:

$$(r_{safedist})^t = -0.1|d_{thresh} - d_{rob}| \quad \text{if } d_{thresh} > d_{rob}, \quad (2.7)$$

where d_{thresh} and d_{rob} denote the *threshold* distance that the robot needs to maintain from an obstacle at any time, and the actual distance that the robot maintains from an obstacle, respectively. For multiple obstacles present in the robot’s vicinity, the penalties in equations 5 and 7 are applied for each obstacle. We set $r_{wp} = 10$, $r_{goal} = 20$, and $r_{collision} = -20$ in our formulation.

2.3.2.4 Training Scenarios

The policy training is carried out in multiple stages to ensure fast convergence of the total accumulated reward. When the robot trains in more complicated training scenarios, we also run the robot in simpler training scenarios simultaneously to ensure that previously learned capabilities are not overwritten. We designed several training scenarios to suit our pedestrian prediction network by including walking pedestrian models in the dynamic scenes in our training. Furthermore, we include dense pedestrian scenarios with sudden changes in the pedestrian motion. Our training scenarios are as follows:

- **Random Goal:** The robot is given a random goal in an empty world, and actions leading the robot towards the goal are rewarded. In this scenario, the partially trained model learns basic goal-reaching capabilities.
- **Dense-Static:** The robot is given a random goal in a world cluttered with static obstacles. During training, the policy augments its previously learned goal-reaching capabilities with basic static obstacle avoidance.

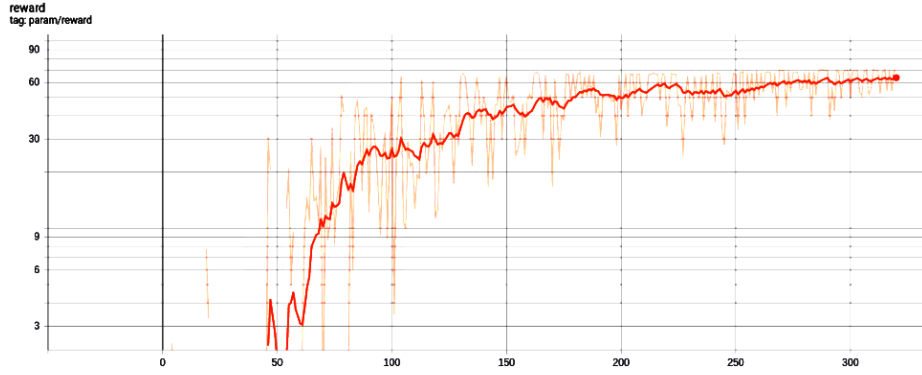


Figure 2.6: Convergence of our reward function (shown for positive reward values) versus the number of iterations. Explicit pedestrian prediction helps our training converge faster and smoother when compared with the case with no prediction.

- **Random-Pedestrians:** The policy from the previous scenario is now trained in a world with randomly walking pedestrians. The pedestrian prediction observations now play a major role in training the policy for dynamic obstacle avoidance. We vary the positions, trajectories, and densities of the pedestrians.
- **Dense-Random-Pedestrians:** In this scenario, the robot needs to navigate through a dense crowd of randomly walking pedestrians before reaching its goal.

2.3.3 Results and Evaluations

In this section, we describe our implementation and highlight DenseCAvoid’s performance in different scenarios. We also compare it with policies trained without trajectory prediction and highlight the benefits of explicitly modeling pedestrian behavior. The convergence of the logarithm of our reward function versus the number of iterations is shown in Fig. 2.5.

Metrics	Sensor Configuration	Dense-Static	Random-Sparse-Ped	Dense-Ped
Success Rate	Depth Camera	0.26	0.73	0.55
	Depth Camera + LiDAR	0.6	0.733	1
	DenseCAvoid	0.93	0.87	1
Avg Trajectory Length	Depth Camera	N/A	5.5	16.3
	Depth Camera + LiDAR	N/A	5.11	15.51
	DenseCAvoid	N/A	6.39	16.87

Table 2.1: Relative performance of our DenseCAvoid hybrid method versus learning-based methods that do not use explicit pedestrian prediction. In the latter category, we use two combinations of sensors (i.e. only depth camera and depth camera + LiDAR). The trajectory length for Dense-Static case is not measured, since we assign the robot’s goals randomly. These numbers highlight the benefit in terms of the success rate (higher is better) of DenseCAvoid. However, the explicit use of prediction can slightly increase the trajectory length (lower is better) to the goal, because the robot may not take the shortest straight-line path.

Metrics	Distance	Depth Camera	Depth camera + LiDAR	DenseCAvoid
Freezing Robot %	< 1.0 meters	100%	100%	100%
	1 - 1.5 meters	53%	33%	5%
	1.5 - 2 meters	27%	0%	0%

Table 2.2: The performance of different methods in avoiding freezing robot scenarios, i.e. the number of instances the robot freezes (lower is better), tested in the Robot Freezing scenario. Our DenseCAvoid method considerably improves the performance when the robot is more than 1m away from an obstacle. In these cases, our explicit prediction of pedestrian trajectory improves the navigation capability and the robot does not freeze.

2.3.3.1 Implementation

Our policy is trained in simulations created using ROS Kinetic and Gazebo 8.6 on a workstation with an Intel Xeon 3.6 GHz processor and an Nvidia GeForce RTX 2080Ti GPU. We use Tensorflow, Keras, and Tensorlayer to create our network. We simulate sensor data using models of the Hokuyo 2-D LiDAR and the Orbbec Astra depth camera in Gazebo during training and testing. The 2-D Hokuyo LiDAR has a maximum range of 4 meters and a field of vision (FOV) of 240°, and provides 512 range values per scan. The Orbbec Astra has a minimum range of 1.4 meters and a maximum sensing range of 5 meters. We use depth images of size 60×80 as inputs to our policy training network. We use a low-resolution image to reduce the latency for processing the

data.

Each pixel value in the depth image varies from 0 to 255. For pedestrian prediction, we normalize the depth images to have values between 1.4 to 5 (corresponding to the camera’s range) before passing it into a pre-trained YOLOv3 and RobustTP network. We observe about 85% prediction accuracy in our dense benchmarks. We convert our depth images into grayscale format before passing it into a pre-trained YOLOv3 and RobustTP network. We mount the same sensors on a Turtlebot 2 robot to test our model in real-world scenarios such as densely crowded corridors with non-smooth pedestrian trajectories. During run-time, we ensure that the rate at which the LiDAR scan data is received, and the rate at which the depth images are processed by RobustTP are synchronized.

2.3.3.2 Testing scenarios

We compare our policy trained with pedestrian prediction with two policies that were trained without pedestrian prediction, but with the same reward function and training scenarios: (i) Policy trained only with depth camera observations, and (ii) Policy trained with LiDAR and depth camera observations. This comparison would clearly highlight the benefits of including pedestrian prediction when all other factors are the same. We consider four different test scenarios that have more challenging static and dynamic scenes, as compared to our training scenarios. This demands tight maneuvers from the robot to reach its goal. The scenarios we consider are:

1. **Dense-Static:** Scenario cluttered with static obstacles, with random goals provided to the policy.
2. **Random-Sparse-Ped:** Scenario where the robot must pass through 10 randomly walking

pedestrians to reach its goal.

3. **Dense-Ped:** Scenario where the robot must move against the direction of 15 walking pedestrians in a narrow corridor to reach its goal.

4. **Robot Freezing:** Several (3 – 4) pedestrians are suddenly spawned at different distances (1 – 2 meters) in front of the robot to simulate the freezing-robot scenario. Most prior benchmarks fail in such cases and we highlight the benefits of explicit trajectory prediction.

2.3.3.3 Performance Benchmarks and Metrics

We use the following metrics to evaluate the performance of different navigation algorithms:

- **Success Rate** - The number of times that the robot arrived at its goal without colliding with an obstacle over the total number of attempts.
- **Average Trajectory Length** - The trajectory length that the robot travels before reaching the goal, calculated as the sum of linear segments over small time intervals over the total number of attempts.
- **Robot Freezing %** - The number of times the robot got stuck or started oscillating indefinitely, while avoiding sudden obstacles over the total number of attempts. A lower value is better.

2.3.3.4 Analysis and Comparison

We present our results in Table 2.1. We observe that DenseCAvoid consistently has a higher success rate as the testing scenarios get more complicated. Using only observations from the depth camera for navigation works well for sparse scenarios, however, performs poorly in dense scenarios. This is mainly due to the low field of vision of the depth camera when compared

to a LiDAR. Using observations from depth camera and a 2-D LiDAR performs slightly better than a single sensor. However, explicit modeling of the pedestrian future trajectory and behavior improves the results in these scenarios. However, the use of trajectory prediction can increase the trajectory length, as the robot may take a larger turn during collision avoidance.

The main benefit of DenseCAvoid arises in terms of dealing with the freezing robot problem, as can be observed from Table 2.2. All methods fail in scenarios where a pedestrian suddenly appears within 1 meter from the robot. This is a consequence of limiting the angular velocity of the robot between $(-0.4, 0.4)$ rad/sec to avoid high oscillations during navigation. Although a higher angular velocity range (say -1 to 1 rad/sec) could avoid such sudden pedestrians, it also leads to undesirable oscillations in some instances. It also prolongs the training time due to the increase in the oscillation penalty earned by the robot. Prior deep reinforcement learning methods can't deal with such situations and the robots tend to freeze. However, DenseCAvoid is able to handle sudden pedestrians about 1 to 1.5 meters away much better than the other methods, leading to oscillations/freezing only 5% of the time. This highlights the benefit of using explicit pedestrian prediction as the classic navigation method along with learning-based methods.

Potential failure scenarios: DenseCAvoid does not allow high-speed maneuvers to avoid pedestrians who could suddenly obstruct (< 1 meter in front of the robot) the robot's path. DenseCAvoid may not perform acute angled turns well or in cases where the data from the depth camera may be corrupted (high infrared interference) from the environment.

2.3.4 Limitations

While DenseCAvoid improves the collision avoidance capabilities for a robot in dense scenarios, it may not work robustly in all possible scenarios. Its accuracy is also limited by the accuracy of trajectory prediction, which is not perfect. The overall performance is mostly governed by the simulated environments used during the training phase and the limitations of the sim-to-real paradigm. Additionally, the actions of DRL policies in general are not easily explainable. Further, the computed actions cannot guarantee collision avoidance, goal-reaching, or reducing the occurrence of FRP. Frozone (section 2.4), our next contributions addresses some of these limitations by formulating a model-based approach to reduce the occurrence of FRP.

2.4 Frozone

In this section, we elaborate on Frozone, an approach that explicitly models spatial regions that could lead to FRP, and calculates a deviation velocity for the robot to avoid such regions. We explain the basic assumptions and notations, and how pedestrians behave in crowded environments in section 2.4.1, then explain how FRP is modeled and avoided in section 2.4.2, and Frozone’s evaluations in 2.4.3. Finally, we discuss some of its limitations in section 2.4.4.

2.4.1 Preliminaries

2.4.1.1 Notation and Symbols

We represent each pedestrian as $[\mathbf{p}^{ped}, \mathbf{u}^{ped}] = [p_x^{ped}, p_y^{ped}, u_x^{ped}, u_y^{ped}] \in \mathbb{R}^4$, where $\mathbf{p}^{ped}$ and $\mathbf{u}^{ped}$ represent the 2-D position and unit vector representing the front/forward direction of the

pedestrian, respectively. We assume that the robot knows its relative goal location and denote it as $\mathbf{g}^{rob}$. All values are specified *with respect to a coordinate frame attached to the robot*, with an origin denoted by $\mathbf{o}^{rob}$ as shown in Fig. 2.7(a). Therefore, the forward heading direction of the robot always corresponds to $(1\hat{i} + 0\hat{j})$ and the direction towards the left is represented as $(0\hat{i} + 1\hat{j})$. We represent scalar values in normal fonts and vectors using bold fonts.

We assume that the robot is equipped with a depth camera to sense nearby pedestrians. Its sensing region $\mathcal{S}^{rob}$ is formulated as a square space with a side length of s_{sen} meters in front of the robot (see Fig. 2.7(a)). Our approach can be easily modified for other sensing regions. The sensing region is offset in front of the robot by f meters to account for sensing inaccuracies in the depth image, which could arise for objects too close to the robot. The depth image I at any time instant t , and the value of a pixel at (i, j) which contains the proximity of an object at that part of the image, are represented as:

$$I^t = \{C \in \mathbb{R}^{w \times h} : f < C_{ij} < f + s_{sen}\}, \quad (2.8)$$

$$1 \leq i \leq w \quad \text{and} \quad 1 \leq j \leq h,$$

where w , h , i and j are the image's width, height, and the indices along the width and height, respectively. C_{ij} is a slight approximation owing to the assumption that the depth camera measures proximity along parallel rays (instead of radial rays) from the camera. We define $dist(\mathbf{a}, \mathbf{b})$ as a function that computes the Euclidean distance between points $\mathbf{a}$ and $\mathbf{b}$.

2.4.1.2 Pedestrian Behavior in Crowds

Pedestrian motions in crowds are influenced by several factors such as crowd density, individual stride length, and need for personal space. Pedestrians tend to walk slower when there is less space in front of them (i.e., dense crowd). The fundamental diagram [62] is used to model this behavior through an inverse relationship between pedestrian velocities and crowd density.

A pedestrian's natural walking speed (v^{ped}) is related to physiological (pedestrian's height and stride length) and psychological (need for personal space) factors [62] using the following equation:

$$v^{ped} = \min \left(\|\vec{v}^{pref}\|, \left(\frac{S\alpha}{H(1+\beta)} \right)^2 \right) \quad (2.9)$$

where $\|\vec{v}^{pref}\|$ is the pedestrian's preferred speed directed towards its goal, which we assume to be 1.3 meters/second (on average). S is the available space in front of the pedestrian, H (height/1.72) is a height normalization factor, and α and β are constants that account for the stride of the pedestrian. We assume that all the pedestrians have the same values for H , α and β . We use the velocity relationship in equation 2.9 to predict the future positions of pedestrians after a time Δt . This formulation is complementary to existing trajectory prediction methods and also accounts for the human psychological need for space while walking. From equation 2.9, we infer that a pedestrian's motion is unaffected when a robot avoids/passes them from behind or if it leaves sufficient space in front of them (high value of S).

Therefore, in our approach, we compute a deviation for the robot velocity such that the robot maintains sufficient distance in-front of the pedestrians, and avoids them from behind

when possible. We define η as the pedestrians' comfortable distance threshold based on social constraints, i.e. the distance that the robot needs to maintain from a pedestrian whenever possible. In addition, we define the *pedestrian-friendliness (PF)* of the robot as,

$$PF = Z * N_{\infty} + \bar{Z} * \min(\text{dist}(\mathbf{o}^{rob}, \mathbf{p}_i^{ped})), \quad (2.10)$$

where Z is a boolean that represents if a pedestrian has been avoided from behind, N_{∞} is a large value, and $\mathbf{p}_i^{ped}$ represents the position of the i^{th} pedestrian in the environment.

To detect pedestrians, we use YOLOv3 [10] as explained in section 2.3.1.1. YOLOv3's outputs are a set of bounding boxes for the detected pedestrians represented as $\mathcal{B} = \{\mathbb{B}_k \mid \mathbb{B} = [\text{top left}, m_{\mathbb{B}}, n_{\mathbb{B}}], \in \mathcal{H}\}$, where top left, $m_{\mathbb{B}}$, and $n_{\mathbb{B}}$ denote the top left corner coordinates, width, and height of the k^{th} bounding box $\mathbb{B}_k$, respectively. $\mathcal{H}$ denotes the set of all pedestrian detections. In addition, it also assigns an ID for a given image and reports the detection confidence.

The detection bounding boxes are used as inputs to DensePeds [112], a pedestrian tracking algorithm that assigns a unique ID to each detected pedestrian across multiple consecutive images. This ID is used to compute each pedestrian's position in the image over time. We use DensePeds as it is robust to the noise in the image and performs with an accuracy of over 93% in dense (> 1 person/ m^2) scenarios.

2.4.2 Methodology

In this section, we describe how we formulate the freezing robot problem. This includes various components including pedestrian classification, the construction of a potential freezing zone, and our formulation to calculate the deviation needed to avoid the freezing zone.

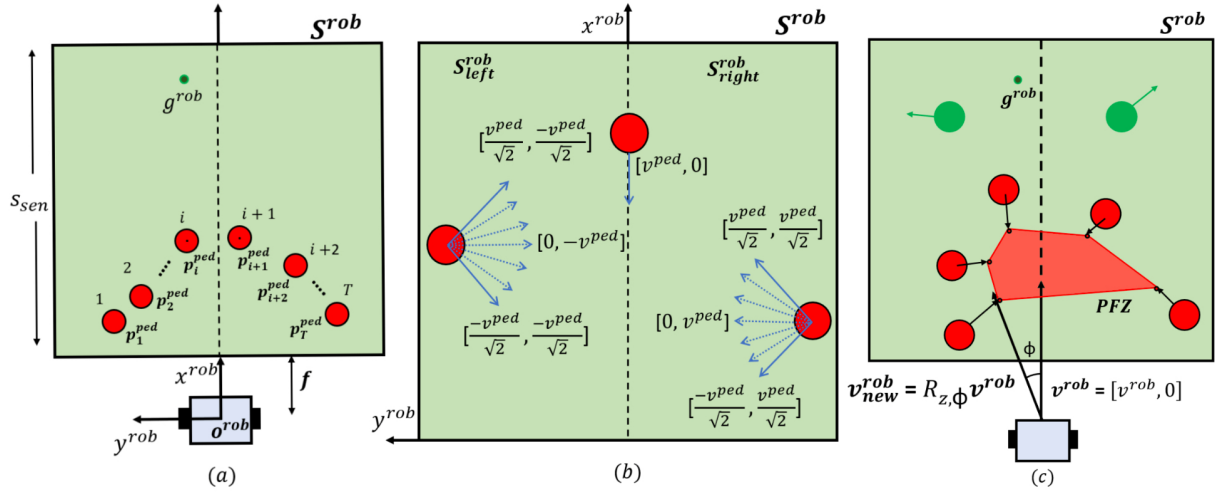


Figure 2.7: **(a)** An example for the freezing robot scenario. The robot does not find enough space between pedestrians (red circles) to move towards its goal and the robot's navigation module concludes that all velocities towards the goal would lead to a collision. **(b)** The range of pedestrian velocity directions that are considered as *potentially-freezing* in the left and right half planes of the sensing region. Pedestrian velocities along the robot's X-axis are special cases of an either head-on approach or a pedestrian moving away. **(c)** The construction of the Potential Freezing Zone (PFZ), which is the convex hull of the predicted positions of *potentially-freezing* pedestrians after time Δt . The deviation angle ϕ is computed such that the robot is directed away from the PFZ with the least amount of deviation from its current velocity.

2.4.2.1 Formulation of FRP

Consider a scenario as shown in Fig. 2.7(a), where a robot faces a set of T pedestrians in its sensing region before reaching its goal. The pedestrians could either be mobile or stationary.

Definition 2.4.2.1 (*Freezing Robot Problem*). If the pedestrians are stationary and are positioned such that,

$$\begin{aligned}
 dist(\mathbf{p}_i^{ped}, \mathbf{p}_{i+1}^{ped}) &< 2\Omega \quad \forall i \in \{1, 2, \dots, T\}, \\
 dist(\mathbf{o}^{rob}, \mathbf{p}_i^{ped}) &\leq \Omega \quad \forall i \in \{1, 2, \dots, T\}, \\
 \mathbf{p}_i^{ped}, \mathbf{p}_{i+1}^{ped} &\in S^{rob},
 \end{aligned} \tag{2.11}$$

where $\mathbf{p}_i^{ped}$ and $\mathbf{p}_{i+1}^{ped}$ are the positions of the i^{th} and $(i+1)^{th}$ pedestrian with respect to the

robot, and Ω denotes the minimum distance threshold that the robot’s collision avoidance module should maintain with all obstacles. If the conditions in 2.11 are satisfied then the planner deems all forward velocities as unsafe. Such a scenario constitutes the *Freezing Robot Problem*. One possible technique (without human cooperation) for the robot to reach its goal is to retrace its path, identify the free space, and re-plan its trajectory. This typically requires global knowledge of the environment, such as a map of the environment and all dynamic obstacles, which may not be readily available. The robot either halts completely or generates undesirable behaviors such as oscillations, which severely degrade the efficiency of the robot’s navigation.

If the humans are non-stationary, the robot disrupts their velocities due to its low proximity in front of them (i.e. low value of S). This reduces the robot navigation’s pedestrian-friendliness. Therefore, we develop an approach where the robot is able to predict if such scenarios could occur in the near future and preemptively avoid them. This simultaneously reduces the occurrence of FRP and improves the robot’s pedestrian-friendliness in terms of social constraints.

2.4.2.2 Computing Pedestrian Poses and Prediction

The pedestrian positions and orientations are obtained by tracking pedestrians in the image produced by the depth camera. Referring to equation 2.8, the value of the $(i, j)^{th}$ pixel in the image C_{ij} contains the proximity of an obstacle present in that part of the image. When a pedestrian k is detected and tracked in consecutive images (Section III.C), the pixel values within the detection bounding box $\mathbb{B}_k$ are averaged to measure the mean distance (d_{avg}) of the pedestrian from the camera. Let the centroid of the bounding box $\mathbb{B}_k$ be denoted as $[x_{\mathbb{B}_k}, y_{\mathbb{B}_k}]$. Then, the

angular displacement ψ of the pedestrian with respect to the robot can be calculated as:

$$\psi = \left(\frac{x_{\mathbb{B}_k}}{w} \right) * FOV_{cam}, \quad (2.12)$$

where FOV_{cam} is the field of view angle of the camera. The pedestrian's position with respect to the robot can be calculated as $[p_x^{ped}, p_y^{ped}] = d_{avg} * [\cos \psi, \sin \psi]$. A pedestrian's orientation unit vectors $[u_x^{ped}, u_y^{ped}]$ can be computed by calculating the difference between the pedestrian positions computed between two consecutive depth images. The pedestrian's perceived space in front of the pedestrian (S) can be trivially calculated based on the computed relative positions of all the pedestrians in the robot's sensing region. Using S, the pedestrian's walking speed v^{ped} is calculated based on equation 2.9.

2.4.2.3 Classifying Potentially Freezing Pedestrians

To predict if the freezing scenario discussed in section 2.4.2.1 could occur, our navigation algorithm first identifies the pedestrians in its sensing region who could cause such freezing behavior and classifies them as *potentially-freezing*. To this end, the sensing region $\mathcal{S}^{rob}$ of the robot is divided equally along the robot's X-axis as $\mathcal{S}_{right}^{rob}$ and $\mathcal{S}_{left}^{rob}$ as shown in Fig. 2.7(b). Consider a pedestrian in the sensing region of the robot with an orientation denoted by $\mathbf{u} = [u_x^{ped}, u_y^{ped}]$. The pedestrian's velocity vector $\mathbf{v}^{ped}$ with respect to the robot's frame, can be obtained by scaling $\mathbf{u}$ by the pedestrian walking speed v^{ped} (from equation 2.9) as

$$\mathbf{v}^{ped} = [v_x^{ped}, v_y^{ped}] = v^{ped} * [u_x^{ped}, u_y^{ped}].$$

Let the robot speed be v^{rob} . Therefore, its velocity vector with respect to its local coordinate frame will be $\mathbf{v}^{rob} = v^{rob} * [1, 0]$. A pedestrian positioned at $\mathbf{p}^{ped} \in \mathcal{S}_{right}^{rob}$ is considered as *potentially-freezing* if its velocity vector $\mathbf{v}^{ped}$ satisfies these constraints:

$$\begin{aligned} v_x^{ped} &\in [-v^{ped}/\sqrt{2}, v^{ped}/\sqrt{2}], \\ v_y^{ped} &\in [v^{ped}/\sqrt{2}, v^{ped}]. \end{aligned} \tag{2.13}$$

Similarly, the velocity vectors $\mathbf{v}^{ped}$ of a pedestrian positioned at $\mathbf{p}^{ped} \in \mathcal{S}_{left}^{rob}$ is considered *potentially-freezing* if it satisfies:

$$\begin{aligned} v_x^{ped} &\in [-v^{ped}/\sqrt{2}, v^{ped}/\sqrt{2}], \\ v_y^{ped} &\in [-v^{ped}, -v^{ped}/\sqrt{2}]. \end{aligned} \tag{2.14}$$

The pedestrian's speed is assumed to be comparable to the robot's speed in both cases ($v^{ped} \sim v^{rob}$).

Proposition 2.4.1. *The distance function ($\text{dist}()$) between any pedestrian with a velocity vector that satisfies the conditions in equations 2.13 or 2.14 and the robot, is a decreasing function with time.*

Proof. Consider $\mathbf{p}^{ped} \in \mathcal{S}_{right}^{rob}$ with a velocity vector $[v_x^{ped}, v_y^{ped}]$ that satisfies equation 2.13. From Fig. 2.7(b), we observe that p_x^{ped} is positive and p_y^{ped} is negative. For simplicity, let us assume that for a time interval $[t_0, t_{fin}]$, $v^{ped} = v^{rob}$. Let $t_{fin} - t_0 = \Delta t$ be a small time interval comparable to the time taken for the depth camera to capture two consecutive images. In this time interval, with respect to the robot, the pedestrian would have moved from (p_x^{ped}, p_y^{ped}) to

$$(p_x^{ped} + v_x^{ped} \Delta t - v^{rob} \Delta t, p_y^{ped} + v_y^{ped} \Delta t). \tag{2.15}$$

We assume no sudden changes in the pedestrian's motion since Δt is small. Since $v^{ped} = v^{rob}$, $v_x^{ped} \in [-v^{ped}/\sqrt{2}, v^{ped}/\sqrt{2}] \implies v_x^{ped} < v^{rob}$ and $v_y^{ped} \in [v^{ped}/\sqrt{2}, v^{ped}] \implies v_y^{ped} > 0$. Then, the distance between the robot and the pedestrian at t_0 is given as,

$$d_0 = \sqrt{(p_x^{ped})^2 + (p_y^{ped})^2}. \quad (2.16)$$

The distance between the robot and the pedestrian at t_f is given as,

$$d_f = \sqrt{(p_x^{ped} + v_x^{ped} \Delta t - v^{rob} \Delta t)^2 + (p_y^{ped} + v_y^{ped} \Delta t)^2}. \quad (2.17)$$

Since $v_x^{ped} \Delta t - v^{rob} \Delta t < 0$ and $|p_y^{ped} + v_y^{ped} \Delta t| < |p_y^{ped}|$, we get $d_f < d_0 \implies \text{dist}(\mathbf{o}^{rob}, \mathbf{p}^{ped})$ is a decreasing function in $[t_0, t_{fin}]$. A similar proof can be formulated for a pedestrian in $\mathcal{S}_{left}^{rob}$. This result implies that the pedestrian velocities satisfying conditions 2.13 and 2.14 move closer the robot, potentially causing freezing. Their motion would also be affected the most by the robot's navigation. Therefore, such pedestrians should be classified as *potentially-freezing*. ■

Pedestrian Classification: The *potentially-freezing* velocity directions for each half of the sensing region are shown in Fig. 2.7(b). If $v^{ped} < v^{rob}$, then the pedestrian is considered as *potentially-freezing*, irrespective of his/her orientation, as the distance between the robot and the pedestrian decreases over time. Pedestrians with velocity vectors of the form $[\pm v^{ped}, 0]$, are cases where the pedestrian is either moving head-on towards the robot or moving away from the robot (maybe at a lower speed than the robot). Such pedestrians are also considered potentially-freezing provided that the pedestrian position is along the X-axis of the robot. All other pedestrians are considered as *non-freezing*. All *potentially-freezing* pedestrians' positions are involved in *freezing* zone

computation.

2.4.2.4 Constructing the Potential Freezing Zone

Definition 2.4.2.4.1 (*Potential Freezing Zone (PFZ)*). PFZ is defined as a conservative region with a high probability for the occurrence of FRP, after a time interval Δt . Using the current position and velocity of a *potentially-freezing* pedestrian i , we predict his/her's position $\hat{\mathbf{p}}_i^{ped}$ after Δt as,

$$\hat{\mathbf{p}}_i^{ped} = \mathbf{p}_i^{ped} + \mathbf{v}_i^{ped} \Delta t \quad i \in 1, 2, \dots, K, \quad (2.18)$$

where K is the number of *potentially-freezing* pedestrians in the sensing zone. With these predicted positions as vertices, a closed region is constructed, as in (Fig. 2.7(c)),

$$PFZ = \text{Convex Hull}(\hat{\mathbf{p}}_i^{ped}), \quad i \in 1, 2, \dots, K. \quad (2.19)$$

We use the convex hull of $\hat{\mathbf{p}}_i^{ped}$ as a conservative approximation because it simplifies the computation of the freezing zone. It also simplifies the deviation angle computation explained in section 2.4.2.5. The cases where $K = 1$ is a special case where the PFZ is a single point. To maintain sufficient distance from the pedestrian in this case, we construct the PFZ as a circle centered around the pedestrian with a fixed radius. The avoidance of the PFZ locally guarantees the prevention of FRP after time Δt . In addition, it improves the robot's pedestrian-friendliness with respect to the K pedestrians, as the robot's deviation away from the PFZ ensures that it does not navigate obtrusively in front of the pedestrians.

2.4.2.5 Calculating Deviation Angle ϕ

At any instant, if $\hat{\mathbf{p}}_i^{ped}$ of the closest *potentially-freezing* pedestrian i positioned at $\mathbf{p}^{ped} = [p_x^{ped}, p_y^{ped}]$ satisfies,

$$dist(\mathbf{v}^{rob} \Delta t, \hat{\mathbf{p}}_i^{ped}) \leq \eta \quad (2.20)$$

(η is the pedestrian comfort distance) and $\mathbf{v}^{rob} \Delta t \in PFZ$, the robot initializes a deviation from its current velocity direction to a new velocity by an angle ϕ as,

$$\mathbf{v}_{new}^{rob} = R_{z,\phi} \mathbf{v}^{rob}, \quad (2.21)$$

where $R_{z,\phi}$ denotes the rotation matrix for an angle ϕ about the robot's Z-axis. This deviation is constrained based on the relative location of the robot's goal and avoids navigating in-front of the pedestrians. ϕ can be computed as,

$$\phi = \min(\phi_1, \phi_2), \quad (2.22)$$

where ϕ_1 and ϕ_2 are given by,

$$\phi_1 = \underset{R_{z,\phi_1} \mathbf{v}^{rob} \Delta t \notin PFZ}{\operatorname{argmin}} \left(dist(R_{z,\phi_1} \mathbf{v}^{rob} \Delta t, \mathbf{g}^{rob}) \right), \quad (2.23)$$

$$\phi_2 = \tan^{-1}(p_y^{ped}/p_x^{ped}), \quad \phi_2 \neq 0. \quad (2.24)$$

If $\min(\phi_1, \phi_2) = \phi_2$, the robot deviates towards the closest pedestrian's current position. Since the pedestrian has a non-zero velocity in the interval Δt , the robot avoids the closest pedestrian from behind. If $\phi_2 = 0$, we use $\phi = \phi_1$, which denotes the least deviation from the goal.

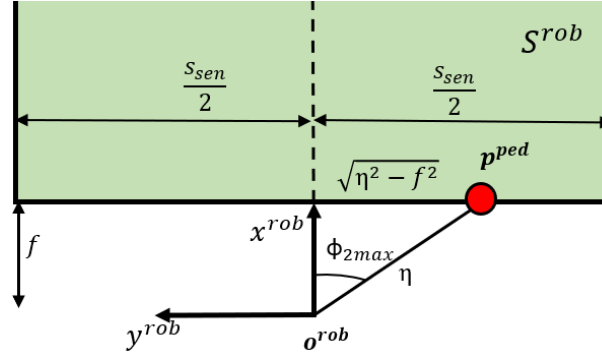


Figure 2.8: The maximum possible deviation ϕ_{2max} occurs, when the closest *potentially-freezing* pedestrian is at $[f, s_{sen}/2]$. This proves that ϕ_2 is always bounded, which implies that $\phi = \min(\phi_1, \phi_2)$ is bounded.

Proposition 2.4.2. *The deviation angle ϕ has an upper bound of $\tan^{-1}(\pm\sqrt{\eta^2 - f^2}/f)$ which depends on η and f .*

Proof. Consider the formulation for ϕ_2 . For the dimensions of the sensing region shown in Fig. 2.7(a), the maximum value of the deviation angle for a certain η could occur if the closest pedestrian is located at $\mathbf{p}^{ped}$ with $p_x^{ped} = f$ meter (just within the sensing region), and $p_y^{ped} = \pm\sqrt{\eta^2 - f^2}$ (see Fig. 2.8). This is because the $\tan^{-1}(p_y^{ped}/p_x^{ped})$ function increases as p_x^{ped} decreases, and f is the least possible value that p_x^{ped} can have. Then, the maximum value of ϕ_2 , $\phi_{2max} = \tan^{-1}(\pm\sqrt{\eta^2 - f^2}/f)$. ϕ_2 is bounded by $\tan^{-1}(\pm\sqrt{\eta^2 - f^2}/f) \implies \phi$ is bounded by $\tan^{-1}(\pm\sqrt{\eta^2 - f^2}/f)$. ■

The constraints on the deviation angle ensure that (i) PFZ can be avoided which reduces freezing and improves pedestrian-friendliness; (ii) since ϕ is the least possible deviation away from the PFZ, the angular motion of the robot is restricted. This results in minimizing the loss of line of sight of the obstacles that are surrounding the robot. We note that our deviation angle computation does not assume pedestrian cooperation for collision avoidance. This formulation is applicable in moderate to dense crowds (≤ 1 person/ m^2), where human cooperation is not required to avoid

freezing.

2.4.2.6 Frozone and Deep Reinforcement Learning

We use a hybrid combination of a DRL-based collision avoidance policy by Long et al [34] and Frozone to compute the robot’s collision-free velocities. The overall system architecture is shown in Fig. 2.9. Frozone modifies the velocities computed by the DRL policy to preemptively avoid potential freezing zones in crowds with lower pedestrian densities (≤ 1 person/ m^2). In dense crowds, the DRL-based policy demonstrates good collision avoidance capability, and the computed velocities are used directly by our hybrid method.

The crowd density in the environment is categorized based on the total number of pedestrians in the robot’s sensing region T . Our switching mechanism for different densities can be expressed as:

$$\mathbf{v}^{rob} = \begin{cases} \mathbf{v}^{Frozone} & \text{if } T \leq s_{sen}^2, \\ \mathbf{v}^{DRL} & \text{if } T > s_{sen}^2. \end{cases} \quad (2.25)$$

$T \leq s_{sen}^2$ corresponds to a density of ≤ 1 person/ m^2 . We also note that Frozone is not restricted to be interfaced with a DRL-based method and can be employed with any collision avoidance scheme.

2.4.3 Results and Evaluations

In this section, we describe Frozone’s implementation and highlight its performance in different evaluation scenarios. We also compare Frozone with prior methods and show significant improvements in terms of navigation performance.

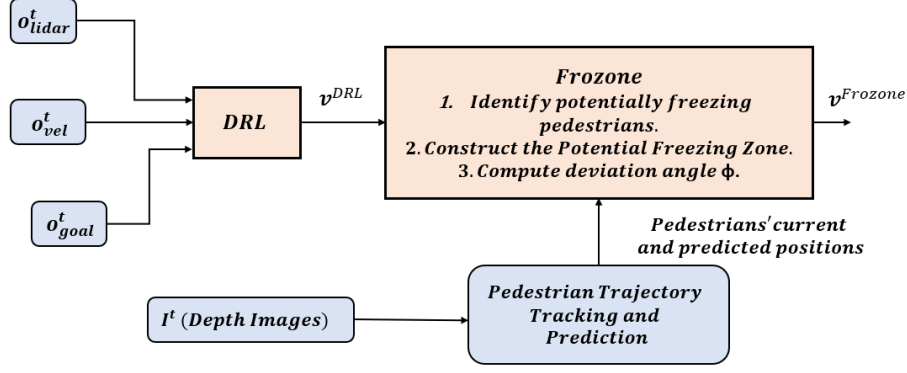


Figure 2.9: Our hybrid method’s system architecture, which uses the velocity provided by a DRL method [34] as a guiding velocity for collision avoidance and modifies it to preemptively avoid Potential Freezing Zones (PFZ). Our method includes 3 main components: 1. Classifying pedestrians as potentially-freezing and non-freezing; 2. Constructing a *Potential Freezing Zone*; 3. Avoiding the PFZ in a pedestrian-friendly manner.

2.4.3.1 Implementation

We first evaluate our method in simulations that were created using ROS Kinetic and Gazebo 8.6. We use a simulated model of a Clearpath Jackal robot attached with models of the Hokuyo 2-D LiDAR and the Orbbec Astra depth camera in Gazebo. The Hokuyo LiDAR, used by the DRL-based method, has a range of 4 meters, an FOV of 240° , and provides 512 range values per scan. The Astra camera has a minimum and maximum sensing range of 0.5 meters and 5 meters, respectively. We use images of size $w \times h = 150 \times 120$ with added Gaussian noise $\mathcal{N}(0, 0.2)$ as inputs to our pedestrian detection and tracking system. For our real-world implementation, we use different cameras; the Astra camera with a Turtlebot 2 and an Intel Realsense depth camera with a Clearpath Jackal robot.

Metrics	Method	Corridor	Crossing	Random-5	Random-10
Success Rate (higher better)	DWA [29]	0.4	0.6	0.6	0.8
	Long et al. [34]	0.1	0.4	0.6	0.2
	DenseCAvoid [53]	0.2	0.6	0.7	0.4
	Frozone + DRL	0.6	0.8	0.6	0.8
Mean Time (lower better)	DWA [29]	32.9	31.9	33.25	32.71
	Long et al. [34]	35.4	39.2	31.3	33.8
	DenseCAvoid [53]	32.03	39.08	32.77	30.04
	Frozone +DRL	31.3	29.06	32.8	32.2
Avg Velocity (higher better)	DWA [29]	0.30	0.31	0.31	0.30
	Long et al. [34]	0.28	0.25	0.30	0.33
	DenseCAvoid [53]	0.33	0.25	0.31	0.33
	Frozone + DRL	0.34	0.35	0.33	0.33

Table 2.3: Comparison of the hybrid combination of Frozone + Long et al’s method’s performance with other learning methods [34, 53], and a traditional collision avoidance method DWA [29] in challenging scenarios. We observe that our hybrid combination’s performance is better or comparable to previous methods. We represent our hybrid combination as Frozone + DRL in the table.

2.4.3.2 Testing Scenarios

We recreate typical indoor and outdoor scenarios that the robot could face in our simulator to evaluate the performance of our algorithm and to compare with prior approaches. For sparse crowds, we assume that the robot takes full responsibility to avoid collisions. As a result, we make no assumptions about human cooperation. We consider the following challenging scenarios to evaluate the performance:

- **Corridor:** The robot must navigate through 15 pedestrians in a straight corridor to reach its goal. The scenario is made more challenging by making the pedestrians walk in multiple pairs or in a zig-zag manner.
- **Crossing:** The robot must move perpendicular to the pedestrian motion in a plus-shaped corridor. The pedestrians may not be sensed until they are very close to the robot.
- **Random-5:** The robot must navigate in a random manner and move through five pedestrians

Metrics	Method	1Ped-3m	1Ped-4m	Ped-perp-3m	Ped-perp-4m
Freezing Rate (lower better)	DWA [29]	100	87	11	0
	Long et al. [34]	100	100	23	11
	DenseCAvoid [53]	100	81	5	0
	Frozone + DRL	0	0	0	0
PF (higher better)	DWA [29]	0.0	0.35	1.02	1.5
	Long et al. [34]	0.0	0.0	0.57	0.95
	DenseCAvoid [53]	0.0	0.24	0.56	1.32
	Frozone + DRL	0.36	0.52	10	10

Table 2.4: Comparison of different methods in terms of freezing rates and pedestrian-friendliness (PF) for challenging 1-person scenarios, when the pedestrian starts at close proximity from the robot and moves at 1 m/s. Our Frozone + Long et al’s method hybrid (represented as Frozone + DRL) outperforms all previous methods, with a significant decrease in freezing and an increase in pedestrian-friendliness in all scenarios.

that result in high pedestrian density in a local region. The maximum pedestrian density in this scenario is 1 person / m^2 .

- **Random-10:** The robot navigates through randomly moving 10 pedestrians. The maximum density however, is < 0.75 person / m^2 . This scenario is used to evaluate the maximum number of pedestrians that each method can handle at any time instant.
- **1 Pedestrian Head-on:** To compare the reduction in the occurrence of freezing and increase in pedestrian-friendliness, we make the robot approach a single pedestrian head-on from different initial positions that are at a distance of 3 meters and 4 meters. The pedestrian moves at 1 m/s towards the robot and halts in-front of the robot, emulating a freezing scenario in dense crowds. This tests the collision avoidance response time of each algorithm, and if the algorithm can avoid freezing.
- **1 pedestrian Perpendicular:** The robot moves perpendicular to a pedestrian’s motion. We evaluate if the robot avoids the pedestrian from front (obtrusive) or back (unobtrusive).

We use the aforementioned single pedestrian scenarios for our evaluation of freezing rates and pedestrian-friendliness, as it provides a more precise way to measure these parameters and observe robot behaviors when compared to more dense scenes.

2.4.3.3 Evaluation Metrics

We highlight the various metrics used to evaluate the algorithm. The mean time and velocity are self-explanatory and correspond to the values when the robot reaches its goal position without a collision.

- **Success Rate** - The number of times that the robot reached its goal without collisions over the total number of attempts.
- **Freezing Rate** - The number of times the robot got stuck or started oscillating for more than 10 seconds while avoiding obstacles over the total number of attempts.
- **Pedestrian friendliness** - We use equation 2.10 to compute the pedestrian friendliness metric with $N_{\infty} = 10$ in our results.

2.4.3.4 Analysis

Table 2.3 shows the results of our comparisons between Frozone + Long et al. [34] DRL method (hybrid) and three previous methods in our simulator: (i) Dynamic Window Approach [29], a traditional collision avoidance method which uses the LiDAR to sense nearby obstacles and forward-simulate the robot’s motion to detect potential collisions, (ii) Long et al. [34], a DRL-based collision avoidance method for dense crowd collision avoidance; (iii) DenseCAvoid [53], a DRL-based method with a trained policy that reduces FRP using pedestrian tracking and

prediction. As mentioned before, our implementation of Frozone is combined with Long et al’s DRL policy [34] for our evaluations.

We observe that Frozone + Long et al’s method has the best success rates of all the methods in all the scenarios highlighted above. It significantly improves the success rates over just using Long et al’s [34] algorithm, and performs better or comparably with DWA. This is because a robot’s success rate is closely tied to avoiding freezing as well. Our hybrid method improves the mean time to goal when compared to Long et al’s method by up to 33%, and the robot’s average velocity increases up to 40 %, while resulting in comparable performances with the other methods.

Freezing and Pedestrian-Friendliness: Frozone significantly reduces the freezing in all our challenging test scenarios. In the scenario where a pedestrian approaches the robot head-on starting from 3 meters away, all previous methods halted to avoid a collision and started oscillating for more than 15 seconds. Frozone + Long et al’s method avoids the circular PFZ constructed around the pedestrian preemptively and prevents freezing (see Fig. 6). When the pedestrian starts from 4 meters away, DWA and DenseCAvoid manage to not freeze in some cases, since the robot has more space and time to react to the pedestrian, and avoid it. Both these methods, however, do not handle dynamic pedestrians in close proximity (< 2 meters away).

Frozone also deviates the robot to avoid pedestrians from behind whenever it results in least deviation from the goal. When the robot moves perpendicular to the pedestrian’s motion, all previous methods take short-sighted actions and try to pass the pedestrian from the front. In some cases, DWA deviated more than 5 meters away from the goal to avoid the pedestrian. Such behavior severely affects both the pedestrian-friendliness and the efficiency of the navigation. In all our trials, Frozone + Long et al’s method passed the pedestrian from behind, without changing

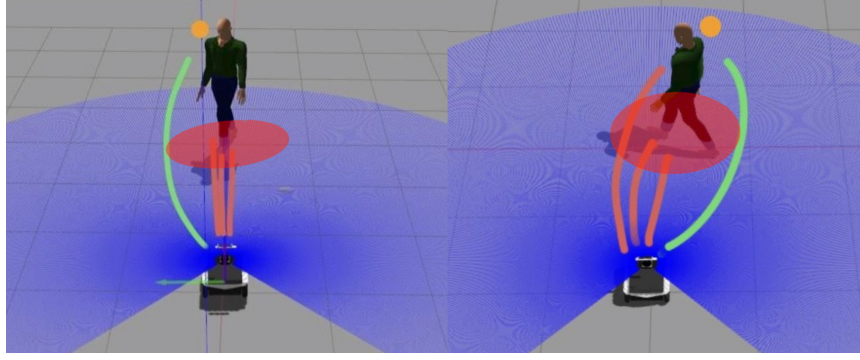


Figure 2.10: The trajectories of our Frozone + Long et al’s method hybrid (in green) compared to the trajectories of previous methods (red trajectories). Our method always avoids the PFZ (red circular region constructed around the pedestrian) and thus avoids freezing. Our method also avoids the pedestrian from behind, thereby not affecting the pedestrian’s motion. The yellow point represents the robot’s goal.

the perceived space in-front of the pedestrian.

In our real-world tests on two differential drive robots, we observed that the robot could switch between the velocities computed by Long et al’s method, and Frozone seamlessly. The performance in real-world scenarios is also simplified as humans generally cooperate with avoiding collisions with the robots. We also test the pedestrian-friendliness improvement in real-world scenes and observe that the robot conservatively navigates behind a pedestrian whenever possible.

2.4.4 Limitations

Frozone’s model-based formulation makes its actions more explainable, and its deviations bounded, unlike DenseCAvoid. While Frozone can reduce the rate of freezing, it cannot avoid it altogether without human cooperation in highly dense crowds. Frozone is formulated for moderately dense crowds. Therefore, its PFZ formulation could be conservative, which could lead to no solutions for deviation in highly dense scenarios.

The behavior of our approach is also governed by the underlying pedestrian tracking algorithm

as well as the techniques used to model pedestrian friendliness. Additionally, Frozone’s FRP condition (equation 2.11) implicitly considers pedestrians in social scenarios as individuals. However, this assumption does not account for pedestrian *groups*, whose members can all be sometimes considered as a single entity.

2.5 Summary

In this chapter, we discussed two of our contributions towards enabling a mobile robot to navigate in crowds of varying densities in indoor environments. Although crowds are challenging in outdoor settings as well, since indoor environments contain more static obstacles that further reduce the available free space for the robot, they are primarily considered in our work. First, we presented DenseCAvoid, a novel collision avoidance method that used a hybrid approach to combine the benefits of learning-based navigation methods with pedestrian trajectory prediction. DenseCAvoid has been implemented in highly dense crowds with pedestrian densities up to 1-2 pedestrian per square meter. Our prediction method does not make any assumptions regarding the motion model for the pedestrian, which results in stable behavior during training and run-time. We validate our work in simulation and in real-world scenarios using a Turtlebot and showed that our approach drastically reduces the freezing robot problem, when pedestrians suddenly appear in front of the robot. We finally discussed its limitations that arise due to DRL.

Secondly, we presented Frozone, a novel method that constructed a spatial zone, where the robot could freeze and be obtrusive to the pedestrians. It then computes an angle to deviate the robot away from this zone. We also combined our method with a DRL-based collision avoidance method to exploit its advantages in dense crowds ($>1-2$ persons/meter²), while reducing the

occurrence of freezing. Frozone overcame some of the limitations of DRL approaches such as unexplainable actions, bounds on the robot's deviation at any time instant, while also using DRL's advantages in higher density crowds.

Both methods have been implemented on real robots such as the Clearpath Jackal and Turtlebot 2 and evaluated in real-world crowds. Both methods share a limitation in that they treat human obstacles as individuals and do not account for the role that groups, their behaviors, and dynamics play in crowds.

Chapter 3: Navigating Amongst Groups in Indoor Environments

3.1 Overview

The contributions of Chapter 2 dealt with dynamic pedestrians as individuals. However, in real-world social scenarios, people typically walk together or interact as groups. The works outlined in this chapter interact with such groups by either identifying and navigating *to* groups of people or recognizing scenarios where groups should be avoided, and determining when it is acceptable to pass through them if necessary.

CS-Robot (COVID Surveillance Robot), the first work described in this chapter was developed during the height of the COVID-19 pandemic to use a robot to identify scenarios where people have breached social distancing norms. We briefly provide an overview of using mobile robots for pandemic response, and to motivate the need for this work.

Global pandemics such as the COVID-19 (COrona VIRus Disease-19) pandemic can cause significant loss of life around the world due to their high infection rates. In such situations, organizations such as the Centers for Disease Control and Prevention (CDC) recommend many guidelines including maintaining social or physical distancing, wearing masks or other facial coverings, and frequent hand washing to reduce the chances of contracting or spreading the virus. Broadly, social distancing refers to the measures taken to reduce the frequency of people coming into contact with others, particularly by maintaining at least 2 meters (6 feet) of physical distance

between individuals. Several groups have simulated the spread of the virus and shown that social distancing can significantly reduce the total number of infection cases [113–118].

Since social distancing is a fundamental method to tackle any pandemic, it is crucial to develop technologies to help detect scenarios where such rules are not being followed, so that appropriate counter-measures can be employed. These actions could also help with collecting data about people who have come in contact with an infected person (contact tracing) [118]. Therefore, developing methods to detect and track groups, and ensuring social distancing is followed could help tackle the next pandemic with relative ease. Additionally, utilizing a mobile robot to attend to scenarios where social distancing has not been followed would also reduce the risk of medical and security personnel in public spaces being infected.

Beyond the pandemic response, detecting groups and identifying their characteristics (e.g. interpersonal relationships) and behaviors can be beneficial for navigating crowded environments. This is because, extensive research in social and behavioral psychology suggests that up to 70% of observed pedestrians in real-world crowds are part of a group [119, 120]. A group is generally regarded as a meso-level concept and corresponds to two or more pedestrians with similar goals over a short or long period of time. As a result, the pedestrians or agents in a group exhibit similar movements or behaviors. Therefore, it is important to understand group characteristics and dynamics to perform socially-compliant robot navigation [42, 54, 64] in complex social scenarios.

One characteristic of groups that could be utilized to improve social-compliant navigation is the social-cohesion or the collective behavior of the group’s members. This is directly linked to the interpersonal relationships between group members. For example, a group of friends or family has higher cohesion than a group of strangers [121, 122]. Cohesion is inversely related to the *permeability* of the group in social settings, i.e. whether another individual can cut through the

group while walking [123]. Many theories have been proposed in psychology and sociology to identify the human behaviors or features that are good indicators of group cohesion. Such features include proximity between group members [124], walking speed [125], group size [123], context or environment [126], etc. Estimating cohesion could help a robot plan a better or more socially compliant trajectory based on the context. For example, in dense crowds (i.e. pedestrian density is more than 1 person/ m^2), the robot could navigate around a group that has high cohesion or the robot could move between members of a group that has low cohesion, similar to how humans navigate in crowded scenarios.

We develop a method to detect social distancing breaches for pandemic response using a mobile robot, and subsequently utilize its group detection capabilities to further detect group cohesion for improved socially-compliant navigation. Our contributions are:

1. CS-Robot, an automatic approach that can use a mobile wheeled robot and wall-mounted CCTV cameras (if available) to detect breaches in social distancing norms. We use a real-time method to estimate distances between people in images captured using an RGB-D camera on the robot and RGB images captured by the CCTV camera. The robot then enforces social distancing by navigating towards the groups of *non-compliant* people, and encouraging them to maintain at least 2 meters of distance by displaying an alert message discreetly on a mounted screen. CS-Robot classifies non-compliant people into different groups, prioritizes attending to them based on whether they are static or dynamic, and computes a goal for the robot to navigate towards the group. It also locks on to a person in a dynamic non-compliant group to more accurately pursue it. We also deploy appropriate measures for privacy protection and de-identification.

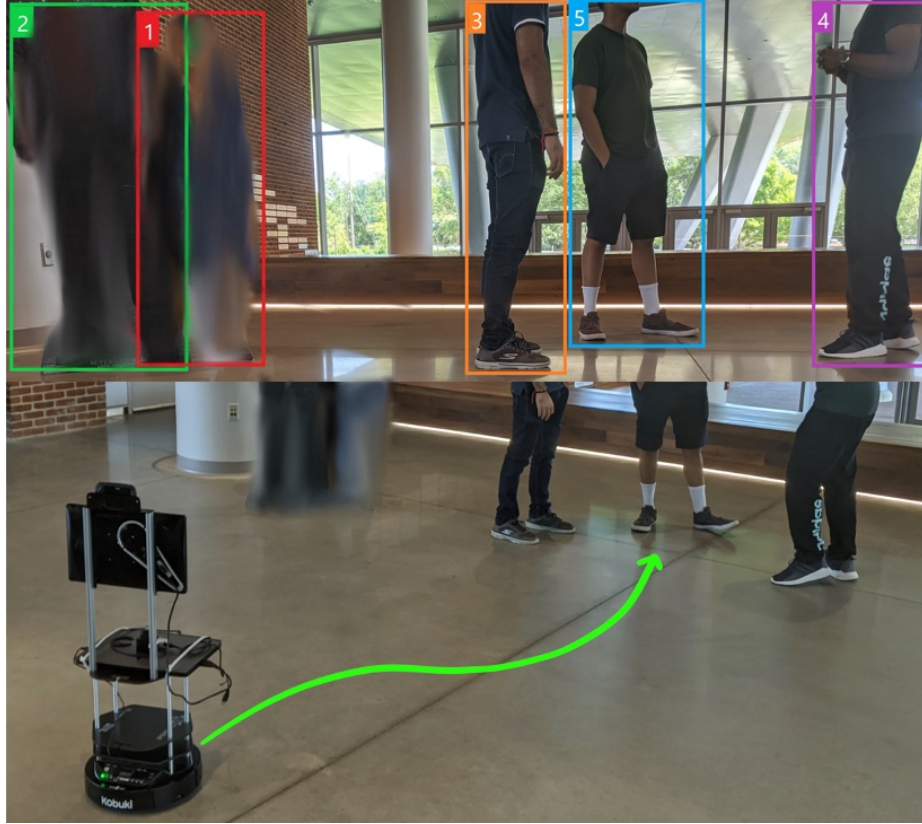


Figure 3.1: Our robot detecting non-compliance to social distancing norms, classifying non-compliant pedestrians into groups and autonomously navigating to the static group with the most people in it (a group with 3 people in this scenario). The robot encourages the non-compliant pedestrians to move apart and maintain at least 2 meters of social distance by displaying a message on the mounted screen. Our CS-robot also captures thermal images of the scene and transmits them to appropriate security/healthcare personnel.

2. CoMet, a novel metric for estimating group cohesion. Our approach is based on social psychology studies and exploits features such as proximity between people, walking speeds, group sizes, and interactions, and uses an RGB-D sensor to detect groups and these visual features. We then present CoMet-based navigation that accounts for group cohesion, while ensuring social-compliance in terms of *naturalness* (section 2.4.3), large deviations and freezing behaviors. Our formulation assumes human cooperation in dense crowds and plans less conservative trajectories than prior methods.

Both methods were implemented on a real Turtlebot robot equipped with a commodity

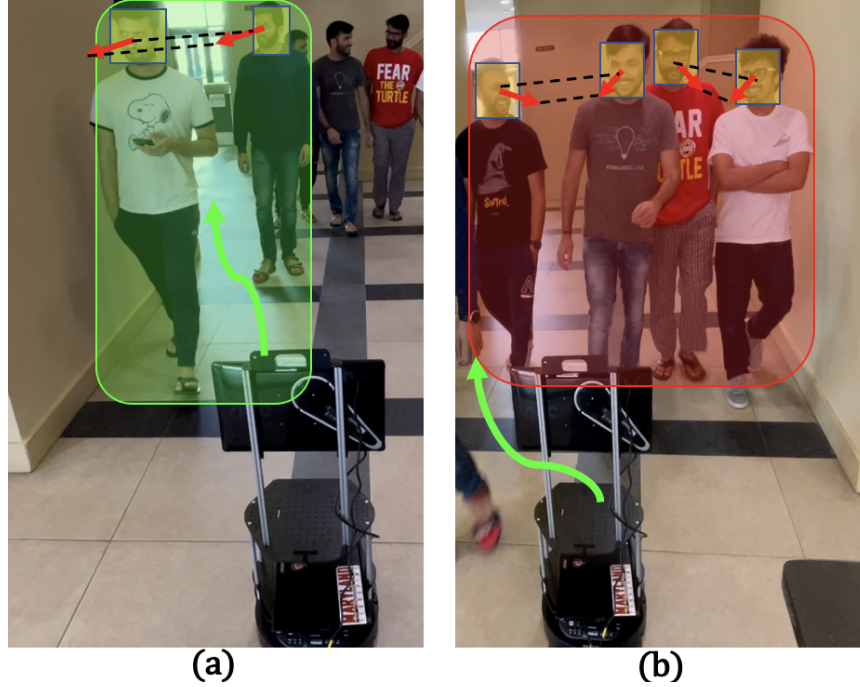


Figure 3.2: Our novel navigation method uses our CoMet metric to compute a collision-free trajectory for a robot in real-world scenarios. CoMet identifies groups in crowds and detects intra-group proximity, walking speed, group size and interactions to estimate a group’s cohesion. (a) In dense scenarios, our navigation algorithm identifies a low-cohesion group (green bounding box) and navigates between its group members (green path) by assuming human cooperation for navigation; (b) Our method detects a high cohesion group (red bounding box) and plans a trajectory around it. Overall, our method improves social-compliance and the naturalness of the trajectory (section 3.4.3.1).

RGB-D sensor and demonstrated improvements in terms of identifying social distancing breaches, and social navigation.

The rest of the chapter is structured as follows. We discuss prior work relevant to detecting groups, social distancing, and group behaviors in section 3.2. Then in sections 3.3 and 3.4, we discuss CS-Robot’s, and CoMet’s methodologies and evaluations in detail, respectively. We conclude this chapter with a summary of the contributions and their limitations.

3.2 Related Work

In this section, we cover the works complementary to CS-Robot in detecting social distancing breaches. Further, we detail the works on detecting groups of people, their behaviors and interactions, that helped formulate CoMet. Apart from these works, the navigation works discussed in Chapter 2 also function as related work to CoMet.

3.2.1 Social Distancing Detection

A comprehensive survey of all the technologies that can be used to detect if social distancing norms are followed properly is given in [118]. This includes a discussion of the pros and cons of technologies such as WiFi (Wireless Fidelity), Zigbee, RFID (Radio Frequency Identification), Cellular, Bluetooth, Computer Vision, AI (Artificial Intelligence), etc. However, many of these technologies require new static, indoor infrastructure such as WiFi routers, Bluetooth modules, central RFID hubs, etc or people to wear detectable tags. Technologies such as WiFi and Bluetooth require people to connect to them using wearable devices or smartphones for tracking. This limits their usage for tracking crowds and social distancing norms in general environments or public places, and may hinder the use of any kind of counter-measures.

Recently, there have also been several vision-based methods that use deep learning [127, 128] to detect social distancing breaches. A large-scale social distancing monitoring system using networked cameras has also been proposed [129]. Solutions based on semi-autonomous and autonomous robots [130] are also becoming popular. For instance, in [131], a quadruped robot with multiple on-board cameras and a 3-D LiDAR is used to enforce social distancing in outdoor crowds using sound-based alerts. CS-Robot (section 3.3) is complementary to these

methods and also helps react to social distancing violations.

3.2.2 Group Detection

Many techniques have been proposed in computer vision for pedestrian and group detection in a crowd. The first step in group detection is to detect individuals in the images or videos. Methods for this step include many deep learning-based approaches for pedestrian detection and tracking [10] and improved methods for high density crowds [132]. These methods have been extended from individual pedestrian detection to group detection [133, 134]. These group-based methods typically use different kinds of clustering based on the proximities between people, their trajectories and their velocities to segregate them into groups [134, 135].

3.2.3 Group Behaviors and Interaction Detection

Different techniques have been proposed for detecting group behaviors and interactions in computer vision and social psychology [126, 136] as well as event identification [137]. Behavior detection and event identification involve the analysis of different features (e.g., collectiveness, stability, uniformity) that represent how people move and interact in a crowd. These include individuals, groups, leaders, followers, etc. They also involve detecting scenarios where groups either merge together or split while walking or running.

Other relevant techniques detect interactions among people in a group based on F-formations [133, 138]. These algorithms estimate features such as people's body and head poses and identify the individuals who are facing each other. Our approach, CoMet (section 3.4), is complimentary to these methods and extends them by using many other features, including proximity, walking,

and interaction, to gauge group cohesion.

Another related domain deals with kinship detection. This class of works deal with training a machine using large-scale datasets [139] to recognize people who are related by analyzing their images and facial features [140, 141]. A detailed survey of such kinship detection methods can be found in [142]. However, these methods require an up-close view of people’s faces, and are not well-suited for working in real-world social settings.

3.3 COVID Surveillance Robot

In this section, we explain what qualifies as a social distancing breach, and how they are detected using a mobile robot and CCTV camera setup. This involves components such as group detection, pedestrians localization. We then explain how CS-Robot locks on to a group member, uses Frozone (section 2.4) to navigate to head towards them to encourage them to maintain a 2-meter distance. We finally present our evaluations of CS-Robot in detecting breaches, and localizing them in section 3.3.2.

3.3.1 Detecting Social Distancing Breaches

In this section, we first explain the architecture of our robot and CCTV setup and the relationship between all the sensors used in the system. We then describe how our method effectively detects a social distancing breach. We refer to people who violate social distancing norms as *non-compliant* pedestrians. We then describe how we classify non-compliant pedestrians into groups and compute the goal for the robot’s navigation for static and dynamic scenarios. Our overall system architecture is shown in Fig 3.3.

3.3.1.1 System architecture

Our overall system architecture is shown in Fig.3.3. Our architecture contains three cameras:

1. RGB-D camera onboard the robot,
2. CCTV camera mounted on the wall (if available), and
3. Thermal camera onboard the robot.

The images from the RGB-D (green arrows) and CCTV cameras (blue dashed arrows in Fig.3.3) are used as inputs to various modules such as pedestrian tracking, homography, and eventually social distance estimation. The data represented by the green arrows is required, whereas the data represented by the blue arrows is optional for the overall operation of the system. The thermal image (red line in Fig.3.3) is analyzed by the pedestrian detection and tracking module, and the outputs are sent only to security or healthcare personnel to monitor symptoms of the tracked people (see section 3.3.1.5). It is not used for social distance estimation or aiding the robot in navigation.

The architecture is designed in a way that the robot can detect social distancing breaches and navigate to non-compliant groups independent of the availability of the CCTV camera using only its onboard RGB-D camera. In the presence of a CCTV camera, the architecture uses its images to more effectively detect non-compliant groups in a larger sensing region with a global perspective and guide the robot.

We detect breach scenarios based on the following criteria. We focus on detecting scenarios where individuals do not maintain a distance of at least 2 meters from others *for a given period of time*. We choose to detect this scenario because it is a fundamental social distancing norm during all stages of a pandemic, even as people begin to use public spaces and restrictions are lifted. An important challenge is to avoid detecting two or more people passing each other as a breach, even if the distance between them is less than 2 meters for a few moments (see Fig 3.4a).

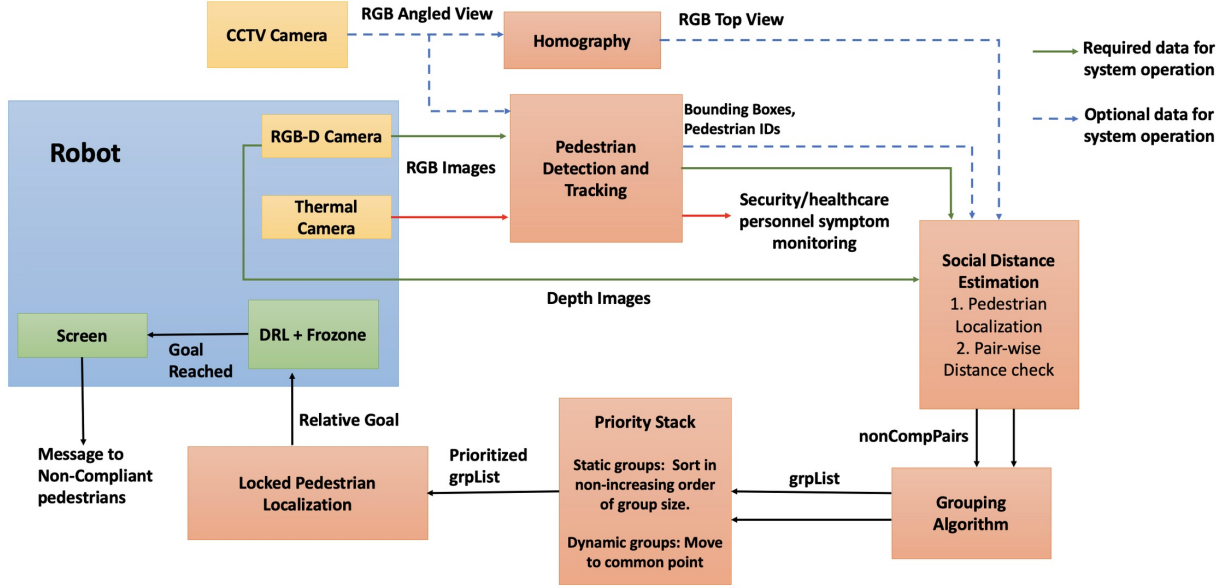


Figure 3.3: Overall architecture of social distance monitoring using CS-robot: The main components include: (i) Pedestrian tracking and localization; (ii) Pairwise distance estimation between pedestrians; (iii) Classifying pedestrians into groups; (iv) Computing a goal for the robot based on whether the group is static or dynamic; (v) Using a hybrid collision avoidance method to navigate towards the goal; (vi) Displaying an alert message to the non-compliant pedestrians to encourage them to move apart; (vii) Thermal image and bounding boxes of detected people are transmitted to security/healthcare personnel.

The robot’s onboard RGB-D camera and the CCTV camera setup (whenever available) continuously monitor the states of individuals within their sensing range. At any instant, breaches could be detected by the robot’s RGB-D camera and/or the CCTV camera.

3.3.1.2 Social distance estimation using RGB-D camera

At any time instant t , we first localize a person detected in the RGB image relative to the robot by using the depth image I^t . The depth image is calibrated to be the same shape as the RGB image. Every pixel in I^t contains the proximity of an object at that location of the image.

First, the detection bounding boxes from the RGB image passed through the pedestrian tracking method are superimposed over the depth image (Fig 3.5). Next, the minimum 10% of

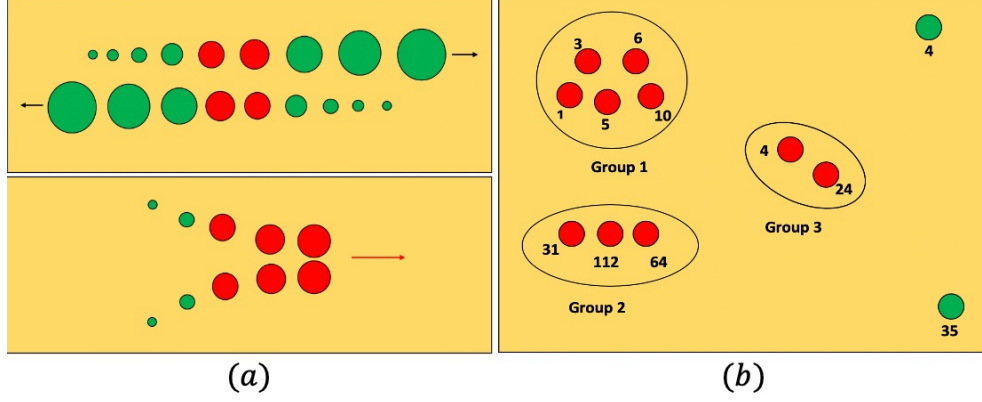


Figure 3.4: **(a):** Our criteria used to detect whether two pedestrians violate the social distance constraint. The pedestrians are represented as circles in two different scenarios. The increasing size of the circles denotes the passage of time. The green circles represent time instants when the pedestrians maintained > 2 meters distance, and the red circles represent instants when they were closer than 2 meters. **Top:** Two pedestrians passing each other. This scenario is not reported as a breach since the duration of the breach is short. **Bottom:** Two pedestrians meeting and walking together. This scenario is reported as a breach of social distancing norms. **(b):** A top-down view of how non-compliant pedestrians (denoted as red circles) are classified into groups. The numbers beside the circles represent the IDs of the pedestrians outputted by Yolov3. The compliant pedestrians (green circles) are not classified into groups as the robot does not have to encourage them to maintain the appropriate social distance.

the pixel values inside the bounding box $\mathbb{B}_P$ are averaged to obtain the mean distance (d_{rob}^P) of a pedestrian P relative to the robot. Denoting the centroid of the bounding box $\mathbb{B}_P$ as $[x_{cen}^{\mathbb{B}_P}, y_{cen}^{\mathbb{B}_P}]$, the angular displacement ψ_{rob}^P of the pedestrian relative to a coordinate frame attached to the robot (with X-axis and Y-axis pointing in the forward and left directions of the robot, respectively) can be computed as:

$$\psi_{rob}^P = \left(\frac{\frac{w}{2} - x_{cen}^{\mathbb{B}_P}}{w} \right) * FOV_{RGBD}. \quad (3.1)$$

Here FOV_{RGBD} is the field of view angle of the RGB-D camera and w is the width of the depth image. We note that $\psi_{rob}^P \in [-FOV_{RGBD}/2, FOV_{RGBD}/2]$. The pedestrian's position with respect to the robot is then calculated as $[x_{rob}^P, y_{rob}^P] = d_{rob}^P * [\cos \psi_{rob}^P, \sin \psi_{rob}^P]$. To estimate the distances between a pair of pedestrians, say P_a and P_b , we use the Euclidean distance function

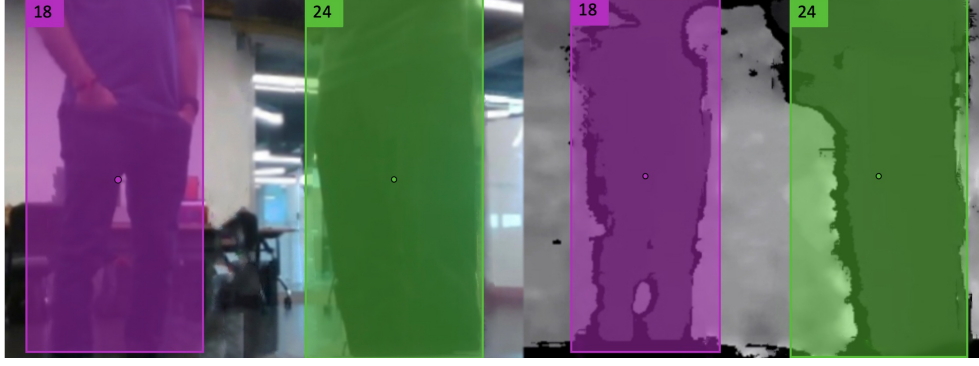


Figure 3.5: **Left:** Two pedestrians detected in the RGB image of the robot’s RGB-D camera with the bounding box centroids marked in pink and green. **Right:** The same bounding boxes superimposed over the depth image from the RGB-D camera. The pedestrians are localized and the distance between them is estimated by the method detailed above.

given by,

$$dist(P_a, P_b) = \sqrt{(x_{rob}^{P_a} - x_{rob}^{P_b})^2 + (y_{rob}^{P_a} - y_{rob}^{P_b})^2}. \quad (3.2)$$

3.3.1.3 Social distance estimation using a CCTV camera

Although the RGB-D camera is mobile, it is limited by a low FOV and sensing range. A breach occurring outside its sensing range will not be reported. Therefore, we utilize an existing CCTV camera setup in indoor settings to widen the scope for detecting breaches. Pedestrian detection and tracking are done as described in section 2.3.1.1. We estimate distances between individuals as follows.

Homography: All CCTV cameras are mounted such that they provide an *angled* view of the ground plane. However, to accurately calculate the distance between any two people on the ground, a top view of the ground plane is preferable. To obtain the top view, we apply a homography transformation to four points on the ground plane in the angled view that form the corners of the maximum-area rectangle that can fit within the FOV of the CCTV camera (see Fig.

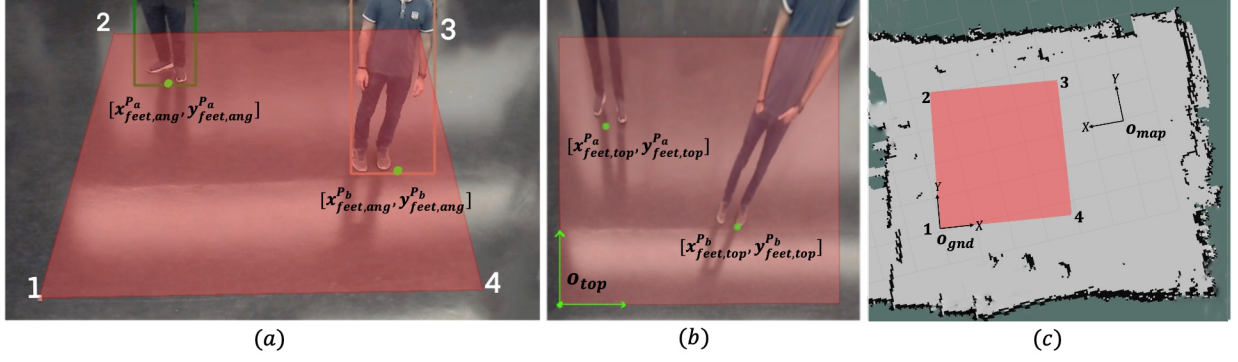


Figure 3.6: **a.** The angled view of the homography rectangle marked in red and corners numbered from the CCTV camera. The green dots mark the points corresponding to a person's feet in this view. **b.** The top view of the homography rectangle after transformation and the origin of the top view coordinate system is marked as o_{top} . The coordinates of the feet points are also transformed using the homography matrix. **c.** A map of the robot's environment with free space denoted in gray and obstacles denoted in black with a coordinate frame at origin o_{map} . The homography rectangle is marked in red and the ground plane coordinate system is shown with the origin o_{gnd} .

3.6a and b) as,

$$[x_{corn,top} \ y_{corn,top} \ 1]^T = M * [x_{corn,ang} \ y_{corn,ang} \ 1]^T. \quad (3.3)$$

Here, $x_{corn,ang}$ and $y_{corn,ang}$ denote the pixel coordinates of one of the four points in the angled CCTV view image. We call the rectangle in the top view, the homography rectangle. $x_{corn,top}$ and $y_{corn,top}$ denote the same point in the transformed top view, and M is the scaled homography matrix which is computed using standard OpenCV (Open Source Computer Vision Library) functions.

Distance estimation between pedestrians: We localize each detected pedestrian within the homography rectangle as follows. We first obtain a point corresponding to the feet of a pedestrian P (denoted as $[x_{feet,ang}^P, y_{feet,ang}^P]$) by averaging the coordinates of the bottom left and the bottom right corners of the bounding box of the pedestrian (see Fig. 3.6a) in the angled CCTV view. This point is then transformed to the top view using equation 3.3 as $[x_{feet,top}^P, y_{feet,top}^P]^T =$

$M * [x_{feet,ang}^P, y_{feet,ang}^P]^T$ (see Fig 3.6b).

The distance between any two pedestrians P_a and P_b is first calculated by using equation 3.2 with the coordinates $[x_{feet,top}^{P_a}, y_{feet,top}^{P_a}]$ and $[x_{feet,top}^{P_b}, y_{feet,top}^{P_b}]$ which is then scaled by an appropriate factor S to obtain the real-world distance between the pedestrians. The scaling factor is found by measuring the number of pixels in the image that constitute 1 meter in the real world.

If the real-world distance estimated from the RGB-D or the CCTV camera images between a pair of pedestrians is less than 2m for a period of time T , a breach is reported for that pair. This process is continued in a pairwise manner and a list of all the pairs of non-compliant pedestrian IDs is then obtained.

3.3.1.4 Enforcing social distancing

Once breaches are detected, either by the RGB-D camera or the CCTV camera or both, the robot must first prioritize them, and compute a goal location in the vicinity of the top-priority breach relative to itself. It must then navigate towards this goal and encourage the non-compliant pedestrians to move apart by displaying an alert message discreetly. We detail the steps involved in the following sections.

Classifying people into groups: In social scenarios, people naturally tend to walk or stand in groups. We define a group as a set of non-compliant people who are closer than 2 meters from each other (see Fig 3.4b). We assume that if the robot reaches the vicinity of a group, it can alert everyone in that group to observe social distancing. In addition, when there are multiple groups of non-compliant people, the robot can prioritize approaching different groups based on whether they are static or dynamic. We classify non-compliant people into groups based on Algorithm 1.

Algorithm 1: Non-Compliant Group Classification.

input : A list nonCompPairs of length S_{input}
output: A list grpList

- 1: nonCompPairs $\leftarrow$ List of pedestrian ID pairs breaching social distancing
- 2: grpList $\leftarrow$ nonCompPairs[0]
- 3: **for** i from 1 to S_{input} **do**
- 4: counter $\leftarrow$ 0
- 5: **for** j from 0 to len(grpList) **do**
- 6: intersection $\leftarrow$ grpList[j] $\cap$ nonCompPairs[i]
- 7: **if** intersection $\neq \emptyset$ **then**
- 8: grpList[j] $\leftarrow$ grpList[j] $\cup$ nonCompPairs[i]
- 9: **else**
- 10: counter $\leftarrow$ counter + 1
- 11: **end if**
- 12: **end for**
- 13: **if** counter == len(grpList) **then**
- 14: grpList.append(nonCompPairs[i])
- 15: **end if**
- 16: **end for**

In Algorithm 1, nonCompPairs is a list that contains the IDs of all the pairs of non-compliant pedestrians obtained in the Breach Detection section. grpList is a list of groups where each group contains the IDs of people who have been assigned to it.

Locked pedestrian To navigate towards a non-compliant group of people, the RGB-D or the CCTV camera must be able to track at least one member of that group to compute a goal in its vicinity. Our method locks on to a person with the least probability of exiting the FOV of either the RGB-D or the CCTV camera (depending on which camera detected the group). This person is called the *locked pedestrian*, and their ID is updated as people's positions change in the cameras' FOVs. The person whose bounding box centroid has the least lateral distance from the center of the image is chosen as the locked pedestrian. The condition is,

$$x_{cen}^{\mathbb{B}_{lp,i}} - \frac{w}{2} = \min_{P \in \mathcal{I}_{\mathcal{G}_i}} x_{cen}^{\mathbb{B}_P} - \frac{w}{2}, \quad (3.4)$$

where $\mathcal{I}_{\mathcal{G}_i}$ is the set of IDs for the detected pedestrians and $\mathbb{B}_{lp_i}$ denotes the bounding box of the locked pedestrian in the i^{th} non-compliant group inside the FOV.

Prioritizing groups in different scenarios: When multiple static and dynamic non-compliant groups are present, our method prioritizes attending to them based on whether the groups are (i) entirely or partially static, or (ii) entirely dynamic.

Entirely or Partially Static Groups: When all or some of the groups within the FOV are static, our method prioritizes attending to each static group by sorting them in the non-increasing order of the number of people in group. Our method then computes a goal to which the robot will navigate based on the locked pedestrian’s location in each group.

If only the RGB-D is in use, the robot localizes the locked pedestrian in the highest priority group relative to itself using equation 3.1). That is, $[x_{goal,rob} \ y_{goal,rob}] = d_{rob}^{lp} * [\cos \psi_{rob}^{lp} \ \sin \psi_{rob}^{lp}]^T$.

Where, $[x_{goal,rob} \ y_{goal,rob}]$ is the goal location relative to the robot, d_{avg}^{lp} and ψ_{rob}^{lp} are the average distance and angular displacement of a locked pedestrian from the robot respectively.

If a CCTV camera is available, the goal computation for the robot requires homogeneous transformations between three coordinate frames: 1. the top-view image obtained after homography, 2. the ground plane, and 3. a map of the environment in which the robot is localized. These three coordinates with origins O_{top} , O_{gnd} and O_{map} respectively are shown in Fig. 3.6b and c. A detailed explanation of our goal calculation can be found in [143]. We provide the final result here:

$$[x_{goal,rob} \ y_{goal,rob}] = [x_{feet,map}^{lp} \ y_{feet,map}^{lp}]^T - [x_{rob,map} \ y_{rob,map}]^T. \quad (3.5)$$

Where $x_{rob,map}$, $y_{rob,map}$ and $x_{feet,map}^{lp}$, $y_{feet,map}^{lp}$ are the X and Y coordinates of the robot and the locked pedestrian’s feet in the map coordinate frame, respectively.

Dynamic Groups: When all the groups in the FOV are dynamic, we use an optimization formulation to compute a goal to attend to the *greatest number of people* before they move away. We first differentiate the groups that are moving closer to and away from the robot by checking if,

$$\begin{aligned} \sqrt{(\hat{x}_{rob}^{lp})^2 + (\hat{y}_{rob}^{lp})^2} &\leq \sqrt{(x_{rob}^{lp})^2 + (y_{rob}^{lp})^2}, \\ [\hat{x}_{rob}^{lp} \ \hat{y}_{rob}^{lp}]^T &= [x_{rob}^{lp} \ y_{rob}^{lp}]^T + \mathbf{v}_{rob}^{lp} \Delta t \end{aligned} \quad (3.6)$$

where $[x_{rob}^{lp} \ y_{rob}^{lp}]$ and $[\hat{x}_{rob}^{lp} \ \hat{y}_{rob}^{lp}]$ denote the current position and position after a small time interval Δt of a locked pedestrian of a group relative to the robot. $\mathbf{v}_{rob}^{lp}$ is the estimated walking vector of the locked pedestrian relative to the robot (as explained in the pedestrian detection section). For all the groups that satisfy this condition, we compute the robot's goal as,

$$[x_{goal,rob} \ y_{goal,rob}] = \underset{x,y}{\operatorname{argmin}} \sum_{i=1}^N \operatorname{dist}([x, y], [\tilde{x}_{rob}^{lp}, \tilde{y}_{rob}^{lp}]) \quad (3.7)$$

where N is the number of groups that satisfy the condition in 3.6. This goal is a common point from which all the groups moving toward the robot can be alerted. We note that this formulation is effective for a small number of groups within the FOV, which we assume is the case when pandemic restrictions are in place. When $N = 1$, the robot actively pursues the group to alert them to maintain social distancing.

Multiple groups, Lawnmower inspection and Alerting: If the same group is detected in both cameras, the goal computed using the CCTV camera is used to guide the robot. The robot uses a display mounted on it to convey a message encouraging social distancing discreetly, once it reaches a group. To improve the effectiveness of the integrated robot and CCTV system in

detecting new non-compliant groups of pedestrians, the robot inspects the blind spots of the CCTV camera continuously by following the well-known lawnmower strategy. In addition, the lawnmower strategy guarantees that 100% of an environment can be covered by navigating to a few fixed waypoints, although it does not guarantee an increase in the number of breaches detected.

3.3.1.5 Thermal Camera and Privacy

As mentioned before, we use the thermal camera exclusively for COVID-19 symptom monitoring. We use our pedestrian detection scheme to detect and track people in the thermal images (see Fig 3.7) and the results are then sent to appropriate security or healthcare personnel who check for COVID-19 symptoms such as a fever. Such a system would be useful in places where people's temperatures are already measured by security/healthcare personnel such as airports, hospitals, etc. Monitoring people's temperatures remotely reduces the risk of security/healthcare personnel contracting the coronavirus while also protecting people's privacy. In addition, we ensure that our algorithms do not use any identifiable information such as facial features.

3.3.1.6 Ethics and privacy

All the evaluations in the following section were conducted based on an IRB (Institutional Review Board) approval by the University of Maryland Institutional Review Board Office (IRB #1608506-1). We followed acceptable privacy preserving guidelines in protecting people's private health information. In particular, we ensure the following: no identifiable information (such as face) of a person is detected, nor stored on any server. Thermal image data is collected only to

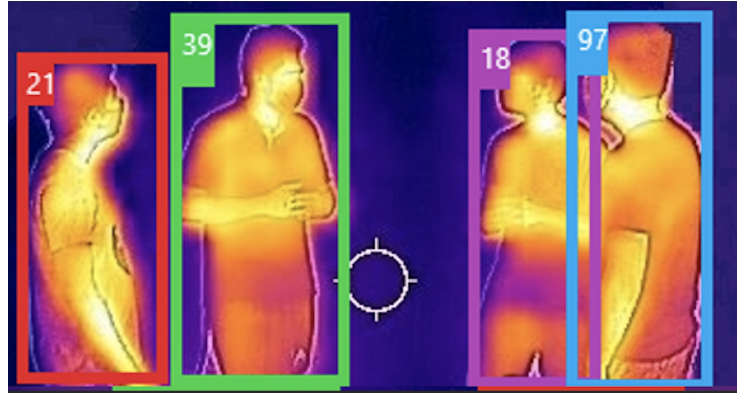


Figure 3.7: Thermal images generated by the thermal camera that is wirelessly transmitted to appropriate security/healthcare personnel. The temperature signatures of the people remain constant irrespective of their orientations. We intentionally have a human in the loop to monitor people’s temperature signatures, and we do not perform any form of facial recognition on people to protect their privacy. Pedestrians are detected on the thermal image to aid the personnel responsible for monitoring the area.

know if a person with symptoms has been in a certain region so appropriate countermeasures can be employed. It is not linked to the person’s age, gender, name, race, ethnicity etc. We also use standard de-identification methods such as visual image redaction for faces, gesture and gait data. The University of Maryland Institutional Review Board Office waived the need for verbal or written consent for our study since we do not map people’s trajectories to any of their personal information. The individuals pictured in Fig.3.1 and 3.7 and Supporting information files (S1 Video) have provided written informed consent (as outlined in PLOS consent form) to publish their image alongside the manuscript.

3.3.2 Results and Evaluations

In this section, we elaborate on CS-Robot’s implementation on a robot, explain the evaluation scenarios and the metrics used to evaluate it, and analyze its effectiveness and limitations.

3.3.2.1 Implementation

We implement CS-Robot on a Turtlebot 2 robot customized with additional aluminium rods to attach a 15-inch screen to display messages to the non-compliant pedestrians. We limit the robot’s maximum linear and angular velocities to 0.65 m/s and 1 rad/s respectively. The pedestrian detection and tracking algorithm is executed on a laptop with an Intel i9 8th generation CPU and an Nvidia RTX2080 GPU mounted on the robot. We use an Intel RealSense (with 70° FOV) RGB-D camera to sense pedestrians and a Hokuyo 2-D LiDAR (240° FOV) to sense other environmental obstacles. To emulate a CCTV camera setup, we used an RGB camera with a 1080p resolution mounted at an elevation of 4 meters, and a sensing region of $4\text{m} \times 4\text{m}$. We use a desktop machine with an Intel Xeon $3.6\text{GHz} \times 8$ processor, Nvidia Titan GPU and 32GB RAM to track pedestrians in the images from the CCTV camera. We use a FLIR C3 thermal camera to generate the temperature signatures of the robot’s surroundings.

3.3.2.2 Evaluation scenarios

For our evaluations, we consider an indoor environment with a wall-mounted CCTV camera with a rectangular sensing region of 4 meters by 4 meters (see Fig.3.6c). The robot is made to operate (either be static, execute lawnmower exploration, or autonomously navigate to a non-compliant group) outside the CCTV’s sensing region. The environment can have static or dynamic obstacles other than humans. We sample the environment uniformly and obtain 40 locations where people can stand while creating scenarios 1 and 2. Twenty of these locations are inside the CCTV’s sensing region and when the robot is static, approximately 10 sampled

locations are within the RGB-D camera's sensing region.

- **Scenario 1:** One person stands in different locations in the sensing range of the RGB-D camera and CCTV camera. The person is allowed to orient themselves in any random direction at the location they stand.
- **Scenario 2:** Two people stand together (< 2 meters distance between them) in one of the 40 uniformly sampled locations in the environment. The people can individually be facing in any arbitrary orientation. They could be standing side by side without occluding each other or one person could occlude the other in various degrees (see Table 3.1). The robot could either be static or executing the lawnmower exploration routine.
- **Scenario 3:** A person walks 5 meters in a direction that is perpendicular to the orientation of the robot. The person walks at different speeds for different trials. The robot is static and is only allowed to rotate in place. The CCTV camera is not used in this scenario.
- **Scenario 4:** A certain number of people walk in random directions in the environment for a period of 60 seconds. The number of people could vary between two and six. They could walk alone or as a group with someone else for the entirety of the 60 seconds. The constraints in this scenario are that the number of non-compliant groups (people walking together) remains constant, and the members of one group do not interact or join another group. The robot executes the lawnmower exploration routine, and the CCTV camera is used to detect breaches.
- **Scenario 5:** Two people walk as a group in a random direction in the environment. Their trajectories could be smooth or have sudden sharp turns. The robot is allowed to be fully

mobile and pursue the walking group. The CCTV is used to detect the group and guide the robot to pursue the non-compliant group.

3.3.2.3 Quantitative metrics

We use the following metrics to evaluate our method.

1. **Accuracy of pedestrian localization:** It refers to the difference between the ground truth location of a pedestrian and the location estimated using our method when using an RGB-D or a CCTV camera.
2. **Number of breaches detected:** The number of locations in an environment (with or without occlusions) at which a social distancing breach can be detected, given a total number of locations uniformly sampled from the environment.
3. **Number of enforcements:** The number of times the robot attended to a breach once it was detected (with or without occlusions).
4. **Tracking duration for a mobile pedestrian:** The time for which the robot can track a walking pedestrian. Longer tracking duration translates to more accurate goal computation and pursuit when dealing with one or more dynamic groups.
5. **Breach detection accuracy:** This metric is defined by the following equation,

$$\text{Breach detection accuracy} = \frac{\sum_{i=1}^T \frac{\text{Number of breaches detected}_i}{\text{Actual number of breaches}_i}}{T}, \quad (3.8)$$

where T is the total number of seconds for which the experiment is conducted, and i is an index denoting every second until T seconds.

Metrics 1 is used for evaluation in scenario 1. Metrics 2, and 3 are used for evaluation in scenario 2. Metric 4 is used for evaluation in scenario 3, and metric 5 is used in scenario 4.

3.3.2.4 Analysis

We elaborate our observations and the inferences drawn from our experiments conducted in the different scenarios previously described, using the aforementioned metrics.

Accuracy of pedestrian localization: In scenario 1, we compare the ground truth locations with the estimated pedestrian location (metric 1) while using 1. the robot’s RGB-D camera, and 2. the CCTV setup. The plots are shown in Fig. 3.8, with the ground truth locations plotted as blue circles and the estimated locations plotted as red circles. The plot in Fig. 3.8a shows the pedestrian being localized relative to the robot coordinate frame (see section 3.3.1). Fig. 3.8b shows a pedestrian being localized in the ground coordinate frame with origin $\mathbf{o}_{gnd}$, which corresponds to corner point 1 of the homography rectangle on the ground (see Fig. 3.6c).

We observe in Fig. 3.8a that when a pedestrian is located closer than 2.5 meters from the robot and along the robot’s X-axis, the localization estimates closely match the ground truth. The difference between ground truth and the estimate increases further away from the robot. This is primarily because the robot localizes a pedestrian using the minimum 10% of the pixels within his/her’s bounding box, whereas the ground truth is measured as a point on the ground.

Furthermore, the orientation of the pedestrian relative to the RGB-D camera also affects the bounding box’s size and centroid, and therefore the localization estimate. However, since the maximum error between the ground truth and estimated values is within 0.3 meters, its effect on the social distance calculation and goal selection for the robot is within an acceptable limit. The

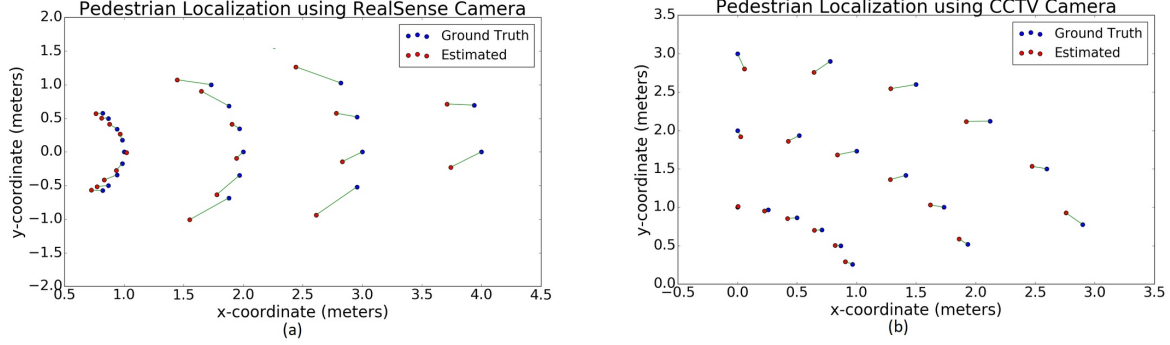


Figure 3.8: Plots of ground truth (blue dots) versus estimated pedestrian localization (red dots) when using the robot’s RealSense camera and the static CCTV camera with more FOVs. **a.** The estimates from the RealSense camera tend to have slightly higher errors because we localize pedestrians using averaged proximity values within their detection bounding boxes, which is affected by the size of the bounding boxes. **b.** Localization using the data from the CCTV camera is more accurate as it tracks a person’s feet. This method is not affected by a person’s orientation. We observe that in both cases, the localization errors are within the acceptable range of 0.3 meters.

accuracy can also be improved with higher FOV depth cameras in the future.

From Fig. 3.8b we see a trend similar to the plot in 3.8a. The farther away a person is from the origin (o_{gnd}), the greater the error between the ground truth and the pedestrian’s estimated location. This is due to the approximations in the homography in obtaining the top view from the angled CCTV view, which carries forward to computing $[x_{feet,top}^{P_a}, y_{feet,top}^{P_a}]$ (see explanations in pedestrian localization using CCTV camera). However, the maximum error between the estimates and ground truths is again within 0.25 meters. Since a pedestrian’s location is estimated by the point corresponding to his/her feet, errors due to the pedestrian’s orientation are less frequent. The average error in the distance estimation between pedestrians is ~ 0.1 meters.

Breach detection and enforcement: In scenario 2, we compare the performance differences in detecting a social distancing breach and enforcing for three configurations: 1. CCTV only,

2. Robot only, and 3. Robot-CCTV hybrid system. Scenario 2 emulates social settings where people could be sitting or standing and interacting with each other in public spaces.

The enforcement capabilities of these systems when people are walking vary extensively. They depend on the initial orientation of the robot (and its sensors) and the people's walking directions and speeds. Therefore, for quantitative comparison, we first standardize our detection and enforcement tests in static settings. We quantify the breach detection capabilities in dynamic settings in the breach detection accuracy section. We qualitatively analyze the detection and pursuit of walking pedestrians in the CCTV-guided walking locked pedestrian pursuit section.

The results are shown in Table 3.1. The CCTV-only configuration is capable of detecting breaches that occur in the 20 sampled locations that are within its FOV. It can also handle occlusions (even $> 50\%$) between pedestrians and detect breaches better due to its global view of the environment. We observed that even with the partial visibility of a person's limbs, the breach detection was accurately performed. The random orientations of the people also do not affect the breach detection's accuracy. We note that this system already improves over current CCTV systems where a human manually has to detect such breaches and initiate countermeasures. However, there is no scope for automatically enforcing social distancing at the location of the breaches.

The robot-only configuration detects fewer breaches than the CCTV setup when the robot is static (due to the RGB-D camera's low FOV). Additionally, occlusions severely affect the robot's number of detections when it is static, especially when a person is occluded by more than 50%. However, when the robot is moving along a lawnmower trajectory outside the CCTV's FOV, it detects the social distancing breaches that could be at any of the 20 locations regardless of occlusions. This is because the robot's RGB-D camera eventually obtains an unoccluded view

Case 1: Static robot no occlusions			
Metric	CCTV-only	Robot-only	Robot-CCTV Hybrid
Number of breaches detected	20	10	30
Number of enforcements	NA	10	30
Case 2: Static robot with 50 % occlusion			
Number of breaches detected	20	7	27
Number of enforcements	NA	7	27
Case 3: Static robot with >50 % occlusion			
Number of breaches detected	20	3	23
Number of enforcements	NA	3	23
Case 4: Lawnmower exploration with 50% occlusions			
Number of breaches detected	20	20	40
Number of enforcements	NA	20	40
Case 5: Lawnmower exploration with >50% occlusions			
Number of breaches detected	20	20	40
Number of enforcements	NA	20	40

Table 3.1: Comparison of three configurations in terms of detecting breaches in social distancing norms when two pedestrians are static in any one of 40 points in a laboratory setting. We observe that CCTV+robot configuration has the most number of breaches detected even when the robot is static and outside the CCTV’s sensing range. When the robot is following lawnmower waypoints outside of the CCTV’s FOV, it can detect a breach in any of the 20 locations that could not be detected by the CCTV camera, even with high levels of occlusion.

as it moves.

We also test the robot-only system’s ability to detect breaches and alert pedestrians in non-laboratory indoor scenes such as lobbies, narrow corridors, etc. These different environments vary in terms of lighting conditions (which affects pedestrian and breach detection) and the available free space for the robot to move. We observe that our system is robust under changes in all the aforementioned factors (see S1 Video).

The robot-CCTV hybrid configuration provides the best performance of the three configurations in terms of detecting novel breaches at most locations when the robot is static. This is because, when the robot is outside the sensing region of the CCTV camera, the hybrid configuration monitors the largest area in the environment. When the robot is mobile, continuously following a lawnmower path, social distancing breaches in all 40 locations can be detected. This configuration also provides better tracking capabilities when a pedestrian is walking (see subsequent sections).

Total number of pedestrians	Number of non-compliant groups at any instant	Breach detection accuracy (for Robot-CCTV Hybrid)
2	1	89%
3 (2 + 1)	1	88%
4 (2 + 2)	2	89.50%
5 (2 + 3)	2	93%
6 (2 + 2 + 2)	3	89.33%
6 (3 + 3)	2	96.50%

Table 3.2: The percentage of breaches detected by the robot-CCTV hybrid setup with different numbers of walking pedestrians. The table shows the effectiveness of the robot-CCTV hybrid system in accurately detecting the actual number of social distancing breaches in the environment over a period of 60 seconds. Each trial is repeated five times and the results are averaged. The first column indicates the total number of pedestrians and how they are split while walking (eg. 2 + 1 indicates two people walking together and one person walking alone). The high breach detection accuracy denotes that our method and setup are effective in detecting the correct number of breaches in the environment.

We also note that, when people are static, the robot attends to 100% of the detected breaches while avoiding all the obstacles in the environment.

Breach detection accuracy with multiple walking pedestrians: We measure the breach detection accuracy (metric 5) of the robot-CCTV hybrid system in scenario 4 with multiple walking pedestrians. The number of walking pedestrians is varied between two to six for different trials of the experiment and in each trial, the number of breaches is kept constant. Each trial is recorded for 60 seconds (i.e., $T = 60$ in equation 3.8) and is repeated five times and the results are averaged. The results are shown in Table 3.2. The numbers in the parenthesis in the first column indicate how the total number of pedestrians is split into non-compliant groups as they walk.

The breach detection accuracy is a measure of the difference between the number of detected breaches versus the actual number of breaches over a period of time. We observe that for our robot-CCTV hybrid, it is always above 88%, indicating that our method accurately detects the actual number of non-compliant groups. We observed that the detection accuracy was typically

affected when the pedestrians moved into blind spots between the sensing regions of the CCTV camera and the robot’s RGB-D camera. Another factor that affected the results was the robot’s pose due to the lawnmower trajectory relative to the pedestrians when they were walking in random directions.

The size of each non-compliant group also affected the overall results (as observed from rows four and six in Table 3.2). Breaches by bigger groups of people are easier to detect. This is because a breach could be recorded even if a subset of all the group members is visible to the cameras. Moreover, such groups also have a lower chance of being completely occluded by objects in the environment. However, the orientations and the walking speeds of the pedestrians did not affect the breach detection accuracy. Overall, we observe that our robot-CCTV hybrid is effective in accurately detecting the actual number of social distancing breaches in the environment.

Computation time and complexity for breach detection: We recorded the computation times for detecting breaches with the increase in the number of pedestrians in the environment (scenario 4). To evaluate the computation times when more people than six people are present, we use human shaped cardboard cutouts (to have up to 13 pedestrians). We compare the computation times to detect breaches on the laptop mounted on the robot and the desktop that processes the CCTV images. The results are shown in Fig.3.9.

We observe that the computation time increases non-linearly with the increase in the number of detected pedestrians. This follows from observing algorithm 1, which has a worst-case time complexity of $\mathcal{O}(n^2)$, where n is the number of detected pedestrians. On average, due to the RGB-D camera’s limited FOV and sensing range, the robot-mounted laptop does not detect more than seven pedestrians simultaneously. The CCTV’s larger sensing region allows tens of pedestrians to

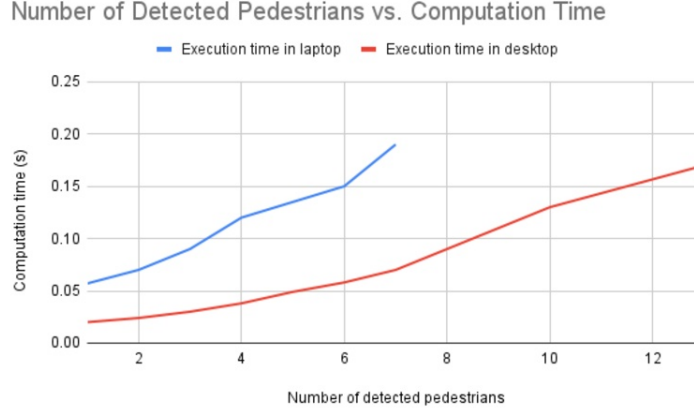


Figure 3.9: The graph shows the computation times while running our breach detection implementations for the RGB-D camera on a robot-mounted laptop, and the wall-mounted CCTV camera on a desktop (see section 3.3.2.1 for specifications). The values were recorded while evaluating in scenario 4 with added human shaped cardboard cutouts to get the total number of detected pedestrians to be 13. We observe that based on the FOV and sensing regions of the two cameras, the corresponding computation times in the laptop and desktop are satisfactory.

be detected. For both these cases, from the graph in Fig.3.9, we observe that the breach detection time is around 0.16 seconds. This is well within the acceptable rate for the robot to enforce social distancing norms even with dynamic pedestrians.

We also observed that since pedestrian and breach detection were not affected by the orientations and velocities of the pedestrians, the time complexity also does not depend on those factors.

RGB-D pedestrian tracking duration In this experiment, we observe the effects of our robot's angular motion on the duration for which it can track walking pedestrians using only its onboard RGB-D sensor with low FOV. We evaluate the pedestrian tracking time metric in scenario 3. We vary the maximum angular velocity of the robot to measure the differences in tracking performance (Table 3.3).

As expected, we observe that the greater the maximum angular velocity of the robot, the

Case 1: Maximum Robot Angular Velocity = 0.5 rad/sec	
Pedestrian Velocity (m/sec)	Tracking time (sec)
0.25	20 (20)
0.5	6.59 (10)
0.75	3.15 (6.67)
1	2.95 (5)
Case 2: Maximum Robot Angular Velocity = 0.75 rad/sec	
0.25	20 (20)
0.5	10 (10)
0.75	3.91 (6.67)
1	2.93 (5)
Case 3: Maximum Robot Angular Velocity = 1.0 rad/sec	
0.25	20 (20)
0.5	10 (10)
0.75	6.58 (6.67)
1	2.77 (5)

Table 3.3: The duration for which the robot tracks a walking pedestrian for different pedestrian walking speeds and maximum angular velocities of the robot. The pedestrian walks 5 meters in a direction perpendicular to the robot’s orientation and it has to rotate and track the walking pedestrian. The ideal time for which a pedestrian should be tracked is given in the bracket beside the actual time. The robot can effectively track a pedestrian walking at up to 0.75 m/sec when its angular velocity is 1 rad/sec.

better it can track a fast-moving pedestrian. However, for high walking speeds ($\geq 0.75m/s$) even a very high maximum angular speed (1 rad/s) is insufficient to track a person for the ideal time that the person needs to be tracked (shown in gray in Table 3.3). This justifies the need for integrating with the existing CCTV camera setup for better tracking capabilities.

CCTV-guided walking locked pedestrian pursuit: We qualitatively demonstrate how our robot pursues two walking non-compliant pedestrians (scenario 5) by plotting their trajectories in the cases where the RGB-D (see Fig.3.10a and b) or the CCTV camera (see Fig.3.10c and d) detects them. Fig. 3.10a shows that when the pedestrians walk in a smooth trajectory without sharp turns, the robot is able to successfully track them throughout their walk.

In Fig.3.10b, we observe that when the pedestrians make a sharp turn and manage to go

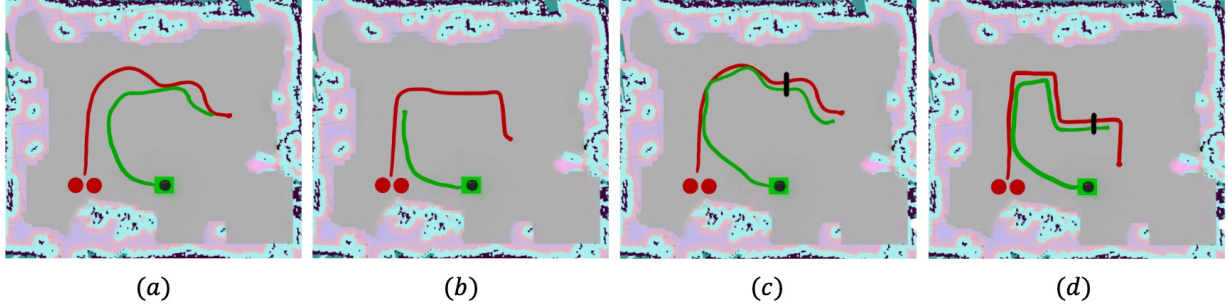


Figure 3.10: Trajectories of two non-compliant pedestrians (in red) and the robot pursuing them (in green) in the mapped environment shown in Fig. 3.6c. The pink and blue colors denote the static obstacles in the environment. **a.** The robot only uses its RGB-D camera to track the pedestrian and pursues the pedestrians successfully when they move in a smooth trajectory. **b.** The robot’s RGB-D camera is unable to track the pedestrians when they make a sudden sharp turn. **c.** When the CCTV camera is used to track the pedestrians, the robot follows their trajectories more closely. **d.** Pedestrians making sharp and sudden turns can also be tracked. The black line denotes the point at which the pedestrians leave the CCTV camera’s FOV, and the RGB-D camera tracks the pedestrians from this point. Sharp turns in **d** again become a challenge.

outside the limited FOV of the RGB-D camera, the robot is unable to pursue him/her. The pedestrians were walking at speeds 0.75 m/sec. This issue is alleviated when the CCTV camera tracks both the pedestrians instead of the RGB-D camera. Fig.3.10c and d show that the robot is able to track the pedestrians more closely and accurately with the goal data computed using the CCTV’s localization. In addition, sudden and sharp turns by the pedestrians are handled with ease, and pedestrians moving at speeds 0.75 m/sec can be tracked and pursued, which was not possible with the robot-only configuration. When the pedestrians move out of the CCTV camera’s FOV (black line in Fig.3.10c and d), the data from the robot’s RGB-D camera helps pursue the two pedestrians immediately. However, the pedestrians’ sharp turns again become a challenge to track.

3.3.3 Limitations

CS-Robot has a few limitations. For instance, it does not distinguish between strangers (low interpersonal closeness), friends/family (high interpersonal closeness). Therefore, *all* individuals in an indoor environment are encouraged to maintain a 6-foot distance from each other. Our current approach for issuing a warning to non-compliant pedestrians using a monitor has limitations, and better human-robot interaction approaches must be explored. As more such monitoring robots are used to check for social distances or collect related data, this could also affect the behavior of pedestrians in different settings. We need to perform more studies on the social impact of such robots.

While exploring human-robot interaction is beyond the scope of this dissertation, we address the problem of estimating the *interpersonal closeness* or *cohesion* in the work presented in the following section.

3.4 CoMet

In this section, we first explain the preliminary physical features (e.g. proximity, walking speed, etc) in social psychology literature that inspired the notion of a group cohesion metric in section 3.4.0.1. We then explain how these features are detected and used to compute group cohesion in section 3.4.2, and how it can be used for navigation in section 3.4.3. We present CoMet’s evaluations and comparisons in section 3.4.4.

3.4.0.1 Preliminaries

In this section, we give an overview of prior work in social psychology, pedestrian tracking and robot navigation that is used in our approach. We also introduce the symbols and notation used in the section.

3.4.0.2 Social Psychology

We use four features based on prior work in social psychology to estimate the cohesion of a group. We give a brief overview of each of these features.

Proximity: Proximity is chosen based on the proxemics principles established by Hall [124]. The underlying theory states that humans have an intimate space, a social and consultative space and a public space when interacting with others. We extend this idea to unstructured social scenarios where people or pedestrians walk, stand, or sit together. In general, humans maintain closer proximity to other people with whom they closely interact (high cohesion). Many techniques have been proposed for simulating pedestrian movement [144] and collision queries [145].

Walking Speed: [125] studied individual and mixed gender groups' walking speeds in a controlled environment and observed significantly slower speeds when people walk with their romantic partners. *Assertion 1:* A slower-than-average walking speed in a group indicates a close relationship between group members (high cohesion).

Group Size: [123] analyzed the perceptions of people when passing through a group of two and four people in a university hallway. It was observed that people tend to penetrate through the 4-person group less than the 2-person group. *Assertion 2:* Humans perceive the

cohesion of a bigger group to be higher (implying lower permeability) than that of a smaller group. Permeability of a group is a measure of the resistance that a moving non-group entity faces while passing between the members of a group.

Interactions: Jointly Focused Interactions (JFI) [146] entail a sense of mutual activity and engagement between people and imply their willingness to focus their attention on others. Therefore, it is chosen as an indicator of cohesion. We extrapolate JFI to signify a higher level of cohesion between group members. Visually, interactions can be detected by estimating peoples' 3-D face vectors [138] and detecting when the vectors point toward each other.

3.4.1 Notations and Definitions

We highlight the symbols and notation used, in Table 3.4. We use i , j , and k to represent indices. All distances, angles, and velocities are measured relative to a rigid coordinate frame attached to the camera (on the robot) used to capture the scene. The X-axis of this frame points outward from the camera and the Y-axis points to the left, with its origin at the center of the image. We use such a representation since our overall approach is local and has no global knowledge of the environment.

The time interval between two consecutive RGB images in the stream is Δt .

Finally, CoMet extends Frozone's (section 2.4) PFZ formulation to groups, making it less restrictive, and suitable for highly dense crowds.

Symbols	Definitions
$\mathbf{p}^{i,t}$	Position vector of person i at time t relative to the robot/camera coordinate frame. $\mathbf{p}^{i,t} = [x^{i,t}, y^{i,t}]$
$\mathbf{v}^{i,t}$	Walking velocity vector of person i at time t relative to the robot/camera coordinate frame. $\mathbf{v}^{i,t} = [\dot{x}^{i,t}, \dot{y}^{i,t}]$
I_{rgb}^t	RGB image captured at time instant t of width w and height h .
I_{depth}^t	Depth image captured at time instant t of width w and height h .
$\mathbb{B}^i$	Bounding box of person with ID i .
$[x_{cen}^{\mathbb{B}^i}, y_{cen}^{\mathbb{B}^i}]$	Centroid of the bounding box $\mathbb{B}^i$.
$\mathbf{x}^{i,t}$	State vector used in Kalman filter for walking vector estimation for person i at time instant t . $\mathbf{x}^{i,t} = [x^{i,t} \ y^{i,t} \ \dot{x}^{i,t} \ \dot{y}^{i,t}]^T$
ID^t	Set of person IDs detected in I_{rgb}^t .
n^k	Number of members in Group $G^{k,t}$.
N	Number of people detected in an RGB image.
$\mathbf{f}_p^i, \mathbf{f}_o^i$	Vectors for person i 's face position and orientation relative to the camera coordinate frame.
K_p, K_w, K_s, K_i	Proportionality and weighing constants for each feature
$CH()$	Convex Hull function with points as its arguments.

Table 3.4: List of symbols used in CoMet and their definitions.

3.4.2 Modeling Group Cohesion

In this section, we present our group cohesion metric, which first classifies pedestrians into groups and then measures their closeness or cohesion. Our method runs in real-time, taking a continuous stream of RGB and depth images as input and detecting the group features highlighted in section 3.4.0.2. Our overall approach based on these features is shown in Fig. 5.3.

3.4.2.1 Detecting Group Features

In this section, we first describe how we track and localize people, detail conditions for a set of people to be classified as a group, and then explain efficient techniques to detect group features from RGB and depth images.

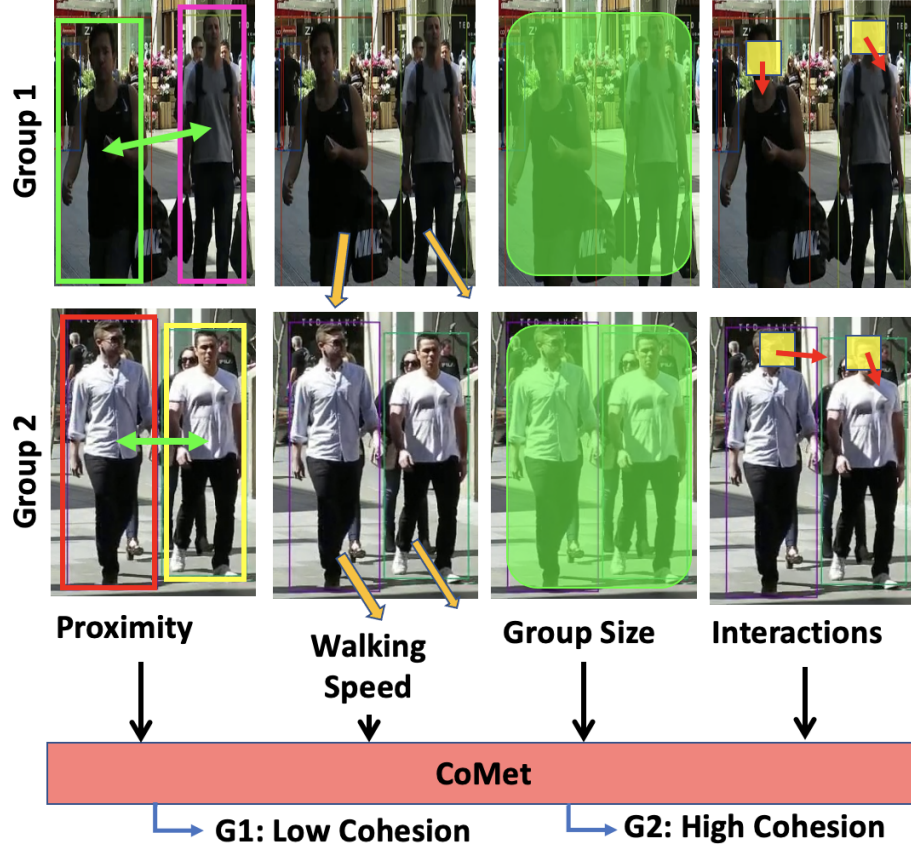


Figure 3.11: Computation of Group Cohesion Metric: We use four features to compute the metric. We highlight how our approach is used on two different groups of pedestrians at a given time instant. These groups are shown in different rows. Group 1 consists of two pedestrians in close proximity, walking away from each other. Their distance is increasing in subsequent frames and they are looking in different directions. This implies little interaction and low group cohesion. Group 2 has two people walking together with their faces oriented towards each other, which indicates high interaction and a high group cohesion.

3.4.2.2 Pedestrian Tracking and Localization

A key issue in detecting the features mentioned in section 3.4.0.2 is to first detect, track, and localize each pedestrian position relative to the camera frame in a continuous stream of RGB images. We use YOLOv5 [147] and Deep Sort [12] algorithms to detect and track people, respectively. YOLOv5 outputs a set of bounding boxes $\mathcal{B} = \{\mathbb{B}^i\}$ for each detected pedestrian i in an RGB image at time instant t (denoted as I_{rgb}^t). $\mathbb{B}^i$ is denoted using its top-left and bottom-right corners in the image-space or pixel coordinates. In addition, we also assign a unique integer

number as an ID for each detected pedestrian.

Next, to accurately localize people, the distance of each detected person relative to the camera coordinate frame must be estimated. To this end, we use a depth image I_{depth}^t , every pixel of which contains the proximity (in meters) of an object at that location of the image. The pixels in I_{depth}^t contain values between a minimum and maximum distance range, which depends on the camera used to capture the image.

3.4.2.3 Group Classification

Let us consider any set of people's IDs $G^{k,t} \subseteq ID^t$. At any time t , if the following conditions hold,

$$\|\mathbf{p}^{i,t} - \mathbf{p}^{j,t}\|_2 \leq \Gamma \quad \forall i, j \in G^{k,t} \quad (3.9)$$

$$\|\mathbf{p}^{i,t} - \mathbf{p}^{j,t}\|_2 \geq \|\mathbf{p}^{i,t} + \mathbf{v}^{i,t} - (\mathbf{p}^{j,t} + \mathbf{v}^{j,t})\|_2. \quad (3.10)$$

then the set $G^{k,t}$ is classified as a *group* in the image I_{rgb}^t . Here, Γ is a distance threshold set manually. The first condition ensures that people are close to each other and the second condition ensures that the group members walk in the same direction. When $\|\mathbf{v}^i\|, \|\mathbf{v}^j\| = 0$ (static groups), only the first condition is used for grouping.

Estimating Proximity: To estimate the proximity between people at time instant t , first the bounding boxes detected in the RGB image by YOLOv5 are superimposed over the depth image. To estimate the distance of a person i from the camera ($d^{i,t}$), the mean of all the pixel values within a small square centered around $[x_{cen}^{\mathbb{B}^i,t}, y_{cen}^{\mathbb{B}^i,t}]$ is computed. The angular displacement $\psi^{i,t}$ of person i relative to the camera can be computed as, $\psi^{i,t} = \left(\frac{\frac{w}{2} - x_{cen}^{\mathbb{B}^i,t}}{w} \right) * FOV_{RGBD}$. Here

FOV_{RGBD} is the field of view of the RGB-D camera. Person i 's location relative to the camera can be computed as $[x^{i,t} \ y^{i,t}] = d^{i,t} * [\cos \psi^{i,t} \ \sin \psi^{i,t}]$.

The distance between a pair of people i and j can then be computed as, $dist(i, j) = \sqrt{(x^{i,t} - x^{j,t})^2 + (y^{i,t} - y^{j,t})^2}$.

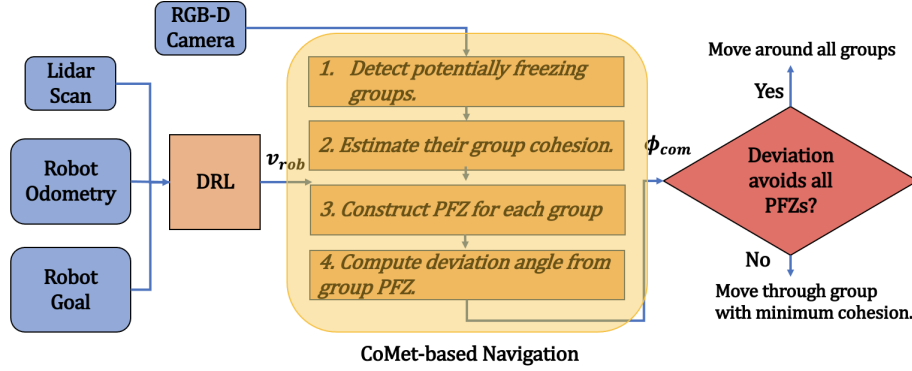


Figure 3.12: CoMet-based socially-compliant navigation. We use a DRL framework used to guide the robot to its goal and handle static obstacles. CoMet-based navigation considers each group in the scene, identifies groups that could result in FRP, constructs group Potential Freezing Zones (PFZ), and computes a deviation angle to avoid such zones. Our formulation results in a lower occurrence of freezing, lower deviations for the robot with respect to pedestrians and groups, and avoiding high cohesion groups by moving around them. In dense scenarios, when there is no feasible solution for the deviation angle, our method navigates the robot through the group with the lowest cohesion. All these behaviors improve the naturalness of the robot trajectory's.

Estimating Walking Speed and Direction: To estimate the i^{th} person's walking vector $\mathbf{v}^i$ in I_{rgb}^t , we use a Kalman filter with a constant velocity motion model. All the detected people in I_{rgb}^t (with their IDs stored in the set ID^t) are modeled using the state vector $\mathbf{x}^t$ defined in Table 3.4. If ID^t contains IDs that were not present in $ID^{t-\Delta t}$, we initialize their corresponding state vectors $\mathbf{x}^t$ with constant values. For all the pedestrians who were detected in previous RGB images, i.e., with previously initialized states, we update their states using the standard Kalman prediction and update the steps [148]. We use a zero mean Gaussian noise with a pre-set variance to model the process and sensing noise.

Estimating Group Size: The size of the group can be trivially computed as the number of IDs in the set $G^{k,t}$.

Detecting Interactions: We use two 3-D vectors to represent the position and orientation of a person's face in I_{rgb}^t . We use OpenFace [149] to localize person i 's face position relative to the camera coordinate frame ($\mathbf{f}_p^i$) on an RGB image and to obtain a unit vector ($\mathbf{f}_o^i$) for the face orientation. Two individuals are considered to be interacting if their face positions and orientations satisfy the following condition:

$$\|\mathbf{f}_p^i - \mathbf{f}_p^j\|_2 > \|\mathbf{f}_p^i + \mathbf{f}_o^i - (\mathbf{f}_p^j + \mathbf{f}_o^j)\|_2. \quad (3.11)$$

This condition checks if the distance between two people's face positions is greater than the distance between the points computed by extrapolating the face orientations (see dashed lines in Fig. 3.2), if they are facing each other. We reasonably assume that non-interacting people do not face each other.

3.4.2.4 Cohesion Components

We now discuss how the detected group features can be used for cohesion estimation. Using multiple features to compute cohesion is advantageous in scenarios where not all features are properly able to be sensed.

Proximity Cohesion Score: We use the average distance between group members to model Hall's proxemics theory as previously extrapolated. As observed in section 3.4.0.2, cohesion between people is inversely proportional to the distance between them. Therefore, the cohesion

score due to proximity is formulated as the reciprocal of the mean distance between group members as

$$C_p(G^{k,t}) = K_p \cdot \frac{n^k}{\sum_{\substack{i,j \in G^{k,t} \\ i \neq j}} dist(i,j)}. \quad (3.12)$$

Walking Speed Cohesion Score Based on the discussion in section 3.4.0.2, we next compare the average walking speeds of each group with the average walking speed of all the detected people in I_{rgb}^t . Therefore, the cohesion score for a walking group ($\|\mathbf{v}^j\| \neq 0 \ \forall j \in G^{k,t}$) due to its walking speed is formulated as

$$C_w(G^{k,t}) = K_w \cdot \left(\frac{\sum_{\forall i \in ID^t} \|\mathbf{v}^i\|}{N} \right) / \left(\frac{\sum_{\forall j \in G^{k,t}} \|\mathbf{v}^j\|}{n^k} \right). \quad (3.13)$$

This reflects assertion 1 made in section 3.4.0.2, since cohesion is inversely proportional to walking speed. The average walking speed of the entire scene is included in this formulation to normalize out the effects of crowding in the scene. If $\sum_{\forall j \in G^{k,t}} \|\mathbf{v}^j\| = 0$, i.e., the group is static, then $C_w(G^{k,t}) = K_w \cdot \eta$, where η is a user-set large constant value.

Group Size Cohesion Score Based on assertion 2 in section 3.4.0.2, the cohesion of a group k is directly proportional to the group size (n^k). Therefore, the group size cohesion score is computed as

$$C_s(G^{k,t}) = K_s \cdot n^k. \quad (3.14)$$

Interaction Cohesion Score The interaction condition between any two people in a group (equation 3.11) can be applied to all pairs in a group, and it's contribution to the cohesion score

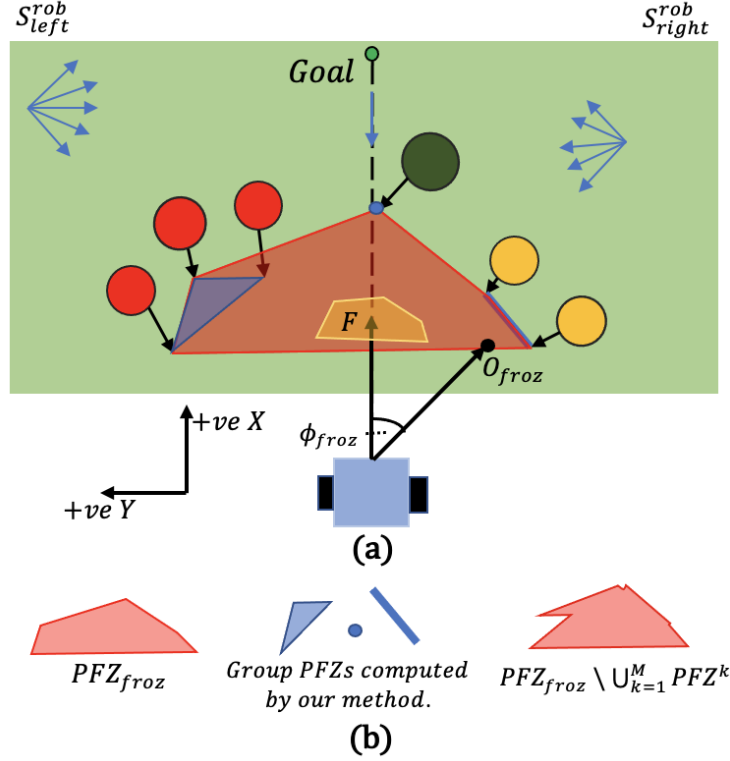


Figure 3.13: **(a)** This scenario shows two groups (red and yellow agents in close proximity) and an individual pedestrian (green) walking towards the robot (blue box). The green rectangle denotes the robot's sensing region and the blue arrows denote the potentially freezing walking directions within each half of the sensing region ($S_{left/right}^{rob}$). **(b) [Left]** PFZ_{froz} (a large 5-sided convex polygon in red) computed by Frozone [54] while considering each individual as a separate obstacle. **[Middle]** Our CoMet-based approach identifies each group and computes the group PFZs (PFZ^k). This corresponds to the blue triangle for the agents in the red group, a line for the agents in the yellow group, a point for the green individual. **[Right]** The region that represents $PFZ_{froz} \setminus \bigcup_{k=1}^M PFZ^k$. These shapes are shown separately to observe the differences. Our proposed method is less conservative and results in a smaller deviation from PFZs (no deviation needed in this case) than Frozone [54]. We also highlight one possible subset of $PFZ_{froz} \setminus \bigcup_{k=1}^M PFZ^k$, which contains positions with deviations $\leq \phi_{froz}$.

of a group can be re-written as

$$C_i(G^{k,t}) = K_i \cdot \frac{1}{n^k} \cdot \sum_{\substack{i \neq j \\ i,j \in G^{k,t}}} \frac{(\theta_{ij})}{\cos \theta_{ij}} \quad \theta_{i,j} \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]. \quad (3.15)$$

Here θ_{ij} is the angle between face orientation vectors $\mathbf{f}_o^i$ and $\mathbf{f}_o^j$ in the X-Y plane of the camera coordinate system. $()$ is the signum function. θ_{ij} is limited to $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ since face orientations are

accurate in this range. Intuitively, we want the cohesion score to be positive and greater than 1, when people are facing towards each other and negative otherwise. Therefore, we choose the ratio $\frac{(\theta)}{\cos \theta}$, as it belongs to the range $[-\sqrt{2}, -1) \cup \{0\} \cup [1, \sqrt{2}]$ when $\theta_{i,j} \in [-\frac{\pi}{4}, \frac{\pi}{4}]$. Since $\cos()$ is an even function, the $()$ function ensures that the formulation is sensitive to the sign of the angle.

3.4.2.5 CoMet: Overall Group Cohesion Metric

Using the individual cohesion scores in equations 3.12, 3.13, 3.14, 3.15, the total cohesion score for a group at time t ($G^{k,t}$) is given as

$$C_{tot}(G^{k,t}) = (C_p + C_w + C_s + C_i). \quad (3.16)$$

Here, $G^{k,t}$ is omitted in the RHS for readability. Note that K_p, K_w, K_s, K_i weigh the different features before adding them to the total cohesion score. If any of the features are not detectable, their contribution to C_{tot} will be zero. This acts as a measure of confidence, as our approach is able to better compute a group's cohesion when more features are detected. Our formulation is not learning-based due to the lack of extensive datasets with annotations of cohesion or related metrics for groups.

Proposition 3.4.1. *The value of the overall cohesion metric $C_{tot}(G^{k,t})$ for a group is bounded.*

Proof. The proof to the proposition follows from the fact that C_p, C_w, C_s, C_i are bounded. The value of $C_p \in (0, K_p \cdot \Gamma]$, since Γ is used as a threshold to group people. The maximum value of C_w is $K_w \cdot \eta$, a large finite constant that is used when a group is static. C_s is bounded above by $K_g \cdot n^k$, which is finite. $C_i \in [-\sqrt{2}K_i, -K_i) \cup [K_i, \sqrt{2}K_i]$. ■

We use these bounds on the cohesion metric to compute appropriate thresholds that are used to categorize groups as low-, medium- and high-cohesion groups.

3.4.3 Cohesion-based Navigation

In this section, we present our socially-compliant navigation algorithm, which uses the group cohesion metric.

3.4.3.1 Socially-Compliant Navigation

Our objective is to improve the naturalness of a robot’s trajectory. We attribute three qualities to natural trajectories: (1) Not suddenly halting or freezing (avoiding FRP), (2) Low deviation angles, (3) Not cutting between high cohesion groups (friends, families etc) in a crowd. This is in accordance with humans’ walking behaviors where people do not suddenly halt or significantly deviate from their goals [33], and do not cut through high cohesion groups while walking [123].

We extend Frozone [54] (section 2.4) by considering groups and their cohesions, and prove that our proposed method leads to smaller deviations from the robot’s goal, and shorter trajectory lengths. It also does not navigate the robot through high cohesion groups. We assume a higher density in the environment (in terms of crowds and static obstacles) than Frozone’s formulation and human cooperation for the robot’s navigation. Frozone prevents the robot from moving in front of a pedestrian to avoid slow down in terms of their walking speeds.

CoMet-based Navigation: To improve the naturalness of a robot’s trajectory, our proposed method includes the following steps: (1) Identifying potentially freezing groups within the sensing

region of the robot and predicting their positions after a time period t_h ; (2) Constructing a PFZ for each group using the predicted future locations of each group member (see blue regions in Fig. 3.13); (3) Computing a deviation angle to avoid the group PFZs while accounting for every group's cohesion. If a feasible solution is not found, the robot navigates between the group with the lowest cohesion in the scene.

Definition 3.4.3.1 (*Potentially Freezing Groups:*) Groups of pedestrians that have a high probability of causing FRP after time t_h . Such groups are identified based on conditions of their average walking direction [54] (see blue arrows in Fig. 3.13). Groups that satisfy these conditions move closer to the robot as time progresses (proven in [54]). We predict the future positions of the potentially freezing group members as

$$\hat{\mathbf{p}}_{pf}^{i,t+t_h} = \mathbf{p}_{pf}^{i,t} + \mathbf{v}_{avg}^{G^{k,t}} t_h, \quad i \in G^{k,t}, \quad k \in \{1, 2, \dots, M\}. \quad (3.17)$$

Here, $\mathbf{v}_{avg}^{G^{k,t}}$ is the average group walking vector, computed as the mean of the walking vectors of the group members and, M is the total number of potentially freezing groups.

Definition 3.4.3.2 (*Group PFZ*) The region in the vicinity of a group where the robot has a high probability of freezing. Instead of constructing the single PFZ as the convex hull of all potentially freezing pedestrians (like in [54]), we construct a PFZ for each potentially freezing group (see Fig. 3.13) as

$$PFZ^k = CH(\hat{\mathbf{p}}_{pf}^{i,t+t_h}), \quad i \in G^{k,t}, \quad k \in \{1, 2, \dots, M\}. \quad (3.18)$$

Every potentially freezing group's PFZ is computed for a future time instant $t + t_h$.

Computing Deviation Angle: If the robot's current trajectory navigates it into any of the group PFZs (implying an occurrence of FRP after time t_h), a deviation angle ϕ_{com} to avoid it is computed. The robot's current velocity $\mathbf{v}_{rob}$ is deviated by ϕ_{com} using a rotation matrix about the Z-axis as,

$$\mathbf{v}'_{rob} = R_{z, \phi_{com}} \cdot \mathbf{v}_{rob}, \quad (3.19)$$

$$\phi_{com} = \underset{\mathbf{v}'_{rob} \cdot t_h \notin PFZ^k}{\operatorname{argmin}} (dist(\mathbf{v}'_{rob} \cdot t_h, \mathbf{g}_{rob})). \quad (3.20)$$

This equation implies that our navigation method deviates the robot by the least amount from its goal such that it does not enter any group's PFZ. However, in dense scenarios, when the robot encounters many potentially freezing groups and their corresponding PFZs, equation 3.20 may not be able to compute a feasible solution for ϕ_{com} . In such cases, a potential solution is to let the robot pass through the PFZ of a low cohesion group (see Fig. 3.15a). In such cases, we formulate the deviation angle as,

$$\phi_{com} = \underset{\mathbf{v}'_{rob} \cdot t_h \in \mathcal{P}}{\operatorname{argmin}} (dist(\mathbf{v}'_{rob} \cdot t_h, \mathbf{g}_{rob})), \quad (3.21)$$

$$\mathcal{P} = PFZ^{min} \setminus (PFZ^{min} \cap PFZ^k) \quad \forall k \in \{1, 2, \dots, M.\}, \quad (3.22)$$

where PFZ^{min} is the PFZ of the group with the minimum cohesion in the scene. Since the permeability of low cohesion groups is high, the above formulation also lowers the probability of freezing.

Proposition 3.4.2. *The deviation angles computed by CoMet-based navigation (equations 3.20 or 3.22), and Frozone (2.22) satisfy the relationship $\phi_{froz} \geq \phi_{com}$.*

Proof. Consider the scenario shown in Fig. 3.13, with PFZ_{froz} depicted as a 5-sided convex

polygon. A vertical line segment connects the robot (O_{rob}) to its goal and the robot's deviation is measured relative to it. Frozone deviates $\mathbf{v}_{rob}$ such that this point lies on the boundary of PFZ_{froz} (equation 2.23) or in the exterior of PFZ_{froz} (equation 2.24), depending on whichever leads to a lower deviation. Let $O_{froz} = [x_{froz}, y_{froz}]$ be the point on the boundary of PFZ_{froz} to which Frozone deviates (by angle ϕ_{froz}) the robot. Based on equations 3.20 and 3.22, our CoMet-based solution deviates the robot to a point in $PFZ_{froz} \setminus \bigcup_{k=1}^M PFZ^k$.

Since, $\bigcup_{k=1}^M PFZ^k \subseteq PFZ_{froz}$, there exists a set $\mathbf{F} \subseteq PFZ_{froz} \setminus \bigcup_{k=1}^M PFZ^k$ such that all positions in $\mathbf{F}$ lead to a deviation angle $\phi \leq \phi_{froz}$. For instance, in Fig. 3.13, $\mathbf{F}$ can be a set just within the boundary of PFZ_{froz} with Y-coordinates greater than y_{froz} . Since equations (3.20) and (3.22) optimize for minimum deviation from the goal, $\phi_{com} \in \mathbf{F}$. This implies that $\phi_{com} \leq \phi_{froz}$. The equality holds when $\mathbf{F} =$ or when the closest edge of PFZ_{froz} to the robot corresponds to the PFZ of a group. Based on the triangle inequality, shorter deviations lead to shorter trajectory lengths. ■

This bound also guarantees that our new navigation algorithm generates trajectory that are more natural than Frozone [54]. We integrate our CoMet-based navigation method with a DRL-based navigation scheme to evaluate it in real-world scenarios [34] and DWA [29] to evaluate it in simulated environments. Figure 3.12) shows the components of our navigation algorithm used to compute trajectories that are more natural in real-world scenes.

3.4.4 Results and Evaluations

In this section, we describe the implementation for computing group cohesion and socially-compliant navigation. We then evaluate CoMet in different standard pedestrian datasets that are

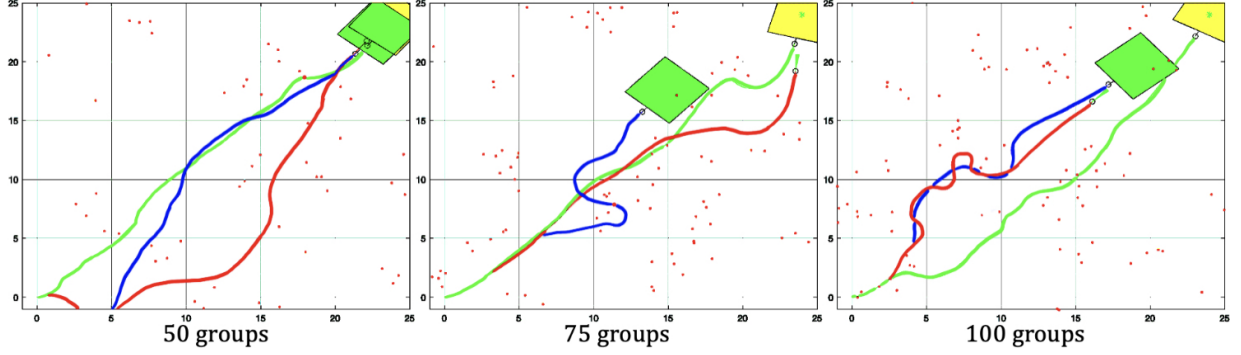


Figure 3.14: Qualitative evaluations of the trajectories generated using our algorithm (green), Frozone [54](blue) and DWA [29] (red) in scenarios with 50, 75 and 100 moving groups (red dots) that are non-cooperative for collision avoidance. The green and yellow squares represent the sensing regions of Frozone and CoMet-based navigation respectively. We observe that our formulation leads to lower deviation angles and more natural trajectories even in dense environments with tens to hundreds of obstacles. Both Frozone and DWA lead to unnatural trajectories with large deviations as the number of obstacles in the environment increases.

annotated with perceived group cohesion levels. We highlight the performance of our navigation algorithm and show the benefits over prior methods in terms of the following quantitative metrics: freezing rate, deviation angle, and normalized path length. We also qualitatively compare trajectories with an increased number of obstacles in the environment.

3.4.4.1 Implementation

In order to evaluate CoMet, we annotate groups in pedestrian datasets such as MOT, KITTI, ETH, etc. as low-, medium- and high-cohesion groups with the help of multiple human annotators. These annotated datasets are used as the ground truth since they reflect how humans perceive the groups in the videos. We choose these datasets because they depict groups in real-world scenarios with various lighting conditions, crowd densities, and occlusions. We use a depth estimation method [150] with RGB images in the datasets to localize different pedestrians in the scene. We manually tune the weighing constants in the CoMet formulation (K_p , K_w , K_s , K_i) and set thresholds on the cohesion score to classify groups into the aforementioned categories based

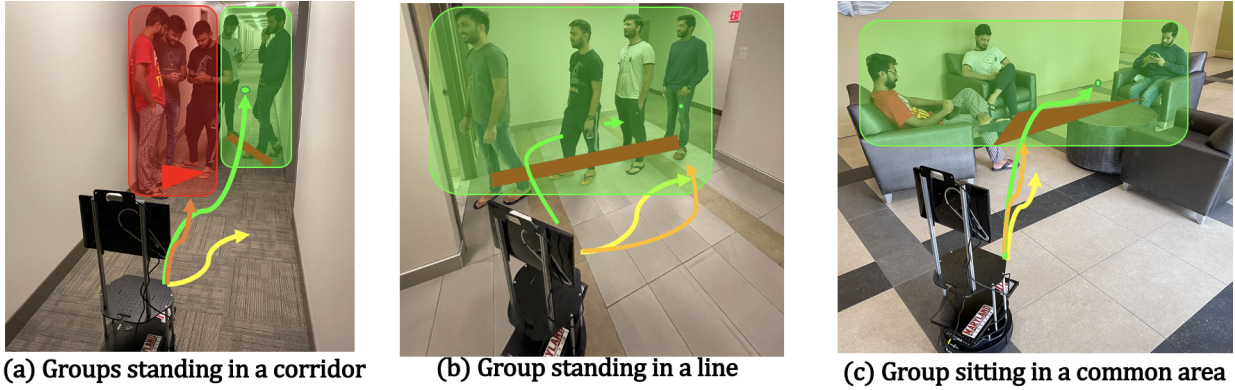


Figure 3.15: Qualitative evaluations of the trajectories generated using our algorithm (shown as green) in different scenarios. We also compare with the trajectories generated using Frozone [54](shown as yellow) and a DRL-based algorithm [34](shown as orange). Each group’s PFZ is shown as a red region on the floor. We evaluate our method in three different real-world scenarios with tight spaces, with people standing or sitting. Our method differentiates between low (in green) and high cohesion (in red) groups, and navigates only between low cohesion groups. Frozone algorithm behaves in a conservative manner and halts the robot in dense scenarios. DRL [34] prioritizes moving towards the goal and passes through high cohesion groups (see (a)). Overall, our approach results in socially-compliant trajectories.

on the annotations in one of the datasets. We evaluate CoMet’s precision and recall in the groups in all other datasets. Since there are no prior methods that compute cohesion, our evaluation is only against the human annotations.

We have evaluated our navigation algorithm using simulations created in MATLAB: (i) with tens to hundreds of groups, each with two to five members (Fig.3.14) and a pre-assigned cohesion metric for each group and (ii) in corridor-like constrained scenarios with tens of pedestrians. The groups move linearly to a goal, and the simulated robot must take full responsibility to avoid collisions with them. We also evaluate our method on a real Turtlebot 2 robot mounted with an Intel RealSense RGB-D camera (for pedestrian tracking and localization) and a 2-D Hokuyo LiDAR (used by the DRL [34] method).

3.4.4.2 Analysis

CoMet Classification: Table 4.4 highlights CoMet’s classification precision and recall in multiple datasets. CoMet’s parameters were tuned to improve its accuracy corresponding to detecting low-cohesion groups, which is required for navigation. This is reflected in the high precision and recall values in the second column. CoMet observes pedestrians in these datasets for ~ 5 seconds. During this period, it is able to update its initial classification as pedestrians’ trajectories change. For instance, a group initially perceived as high-cohesion may have its members move apart and is thereby classified as a low-cohesion group. Moreover, it is easier to detect features corresponding to proximity and walking speed than interactions between pedestrians. This sometimes results in CoMet misclassifying medium- and high-cohesion groups in certain datasets. An interesting observation is that human annotators tend to classify groups in extremes, i.e. as either high-cohesion or low-cohesion groups. This leads to a low number of data points for medium-cohesion groups. This ground truth observation affects the effectiveness of our approach.

Socially-Compliant Navigation: We first qualitatively compare the trajectories (Fig.3.14) of DWA [29] (in red), a DWA-Frozone hybrid [54](in blue), and DWA and CoMet-based navigation (in green) in simulated environments with non-cooperative walking groups of obstacles. We observe that for the same set of dynamic obstacles, our approach computes smaller deviations while also avoiding collisions. DWA and the DWA-Frozone hybrids lead to conservative, and highly sub-optimal deviations from the goal direction. Although the exact set of obstacles each robot faces could be different depending on the trajectories they take, we observe that our method’s deviations at any instant never exceed the deviations of the other two methods in comparison.

Dataset Video	Low-Cohesion	Medium-Cohesion	High-Cohesion
ADL-Rundle	1.00/1.00	-	1.00/0.75
AVG-TownCenter	1.00/1.00	0.50/0.67	0.50/0.33
ETH Jelmoli	1.00/1.00	-	1.00/0.80
ETH Bahnhof	1.00/1.00	1.00/0.67	0.80/1.00
KITTI-16	1.00/1.00	1.00/0.50	0.33/1.00
KITTI-17	1.00/1.00	-	1.00/0.667
MOT17-08	1.00/1.00	-	1.00/0.50
MOT17-11	1.00/1.00	0.75/1.00	1.00/0.833
MOT20-02	1.00/1.00	0.34/0.33	0.50/0.33
TUD	1.00/0.857	-	1.00/1.00
Venice	0.875/0.875	-	0.938/0.60

Table 3.5: This table shows the precision (first value in each column) and recall (second) values for the three classes when CoMet observes groups in the video for ~ 5 seconds. CoMet’s parameters have been tuned based on the ground truth in one of the datasets. We observe good accuracy for high-cohesion and low-cohesion groups. The original datasets have fewer occurrences of medium-cohesion groups, which impacts our precision.

These results reinforce proposition 3.4.2.

We also quantitatively compare the aforementioned three methods in environments with varying numbers (10 – 50) of pedestrians in a corridor-like scenario (which constricts the free space available to the robot). Pedestrians are given random initial locations and velocities, based on which they are classified into groups and PFZ_{froz} and group PFZs (PFZ^k) are computed. The robot needs to navigate through the pedestrians to reach its goal. We use the following metrics: (1) *Average deviation angle* measured relative to the line connecting the start and goal locations; (2) *Freezing Rate* measured as the number of times the robot halted/froze over the total number of trials; and (3) *Normalized Path Length* measured as the robot’s path length over the length of the line connecting the start and goal locations.

Our method results in lower values with respect to all these metrics, as compared to DWA and the DWA-Frozone hybrid in the same scenarios. As the crowd size, density, or number of groups increases, Frozone’s conservative formulation makes the robot freeze at a high rate. This is because Frozone forbids the robot from avoiding a robot from in front of the person, to improve

Metrics	Method	10 peds	20 peds	30 peds	40 peds	50 peds
Avg. Deviation Angle (lower better)	DWA	43.80	45.25	41.78	43.64	49.88
	DWA + Frozone	46.41	46.67	42.85	50.80	53.22
	DWA + Our Method	41.07	44.31	41.11	37.47	34.19
Freezing Rate (lower better)	DWA	0	0.37	0.39	0.36	0.44
	DWA + Frozone	0	0.29	0.25	0.43	0.57
	DWA + Our Method	0	0	0	0	0
Normalized Path Length (lower better)	DWA	1.33	1.31	1.42	1.47	1.49
	DWA + Frozone	1.46	1.45	1.55	1.59	1.51
	DWA + Our Method	1.12	1.29	1.27	1.35	1.35

Table 3.6: We compare different navigation methods (DWA [29], DWA-Frozone hybrid [54] and DWA-CoMet-based navigation algorithm based on three metrics. We observe that our method consistently results in lower values corresponding to all these metrics. This signifies improved naturalness of the robot’s trajectory computed using our approach.

pedestrian-friendliness in low- to medium-density scenes. On the other hand, we observe that our method produces trajectories with high social compliance and naturalness by reducing the occurrence of freezing behavior.

Real-world Evaluations: We qualitatively compare our method’s trajectories with Frozone’s and a DRL algorithm’s [34] trajectories. We highlight the differences in Fig. 3.2. Our approach is able to identify low-cohesion groups successfully and navigate through them without interfering with high-cohesion groups. In contrast, Frozone halts the robot completely, since it does not assume pedestrian cooperation in its formulation. The DRL method prioritizes reaching the goal with the minimum path length and therefore navigates through groups regardless of their cohesion. Therefore, the DRL algorithm can generate obtrusive trajectories.

3.4.4.3 Other Improvements using CoMet

Existing high-fidelity simulators for training Deep Reinforcement Learning methods for navigation simulate dynamic pedestrians as individual obstacles [151]. Large-scale simulators [152] use well-known motion models for individual pedestrians, and do not take into account model group behaviors. CoMet can be used to reverse engineer and simulate group behaviors

based on different cohesion scores. For instance, simulated groups with high cohesion can have low intra-group member proximities, lower than average walking speeds, and larger group sizes. In 3-D simulators with human models, interactions can be modeled based on body orientations.

From an HRI standpoint, in social-distance monitoring robots [143], CoMet could help predict group properties such as interpersonal relationships between members based on their cohesions. This would help identify the groups that need to be issued warnings regarding maintaining social distancing. This leads to more apt interactions between the robot and the humans.

3.4.5 Limitations

CS-Robot groups people only based on their proximity, which in certain cases (e.g. individuals standing in a line) could be erroneous. Conversely, CoMet estimates a group's interpersonal closeness visually, which helps to make improved decisions on *how* to navigate in their presence.

CoMet models cohesion through a linear relationship between the features, which may lead to inaccuracies in some scenarios. In addition, there are other characteristics used to estimate cohesion, including age, gender, environmental context, cultural factors, etc that we do not consider. Our approach also depends on the accuracy of how these features are detected, which may be affected due to lighting conditions and occlusions. Our navigation assumes that different groups exhibit varying levels of cohesions, which may not hold all the time. For instance, in a dense scenario where all groups have high cohesions, CoMet-based navigation may lead to freezing without the cooperation of pedestrians.

3.5 Summary

In this chapter, we presented two methods to navigate a robot by identifying *groups* of people, and measuring their interpersonal closeness using a metric called CoMet.

First, we presented a method to detect social distancing breaches in indoor scenes during a pandemic using simple sensors such as RGB-D and CCTV cameras. We used a mobile robot to attend to the individuals who are non-compliant with the social distancing norms and to encourage them to move apart by displaying a message on a screen mounted on the robot. We demonstrated our method’s effectiveness in localizing pedestrians, detecting breaches, and pursuing walking pedestrians. We demonstrated that the CCTV+robot hybrid configuration outperforms configurations in which only one of the two components is used for tracking and alerting non-compliant pedestrians.

Secondly, we presented a method to compute the cohesion of a group of people in a crowd using visual features. We used our cohesion metric to design a novel robot navigation algorithm that results in socially-compliant trajectories. We highlighted the benefits over previous algorithms in terms of the following metrics: reduced freezing, deviation angles, and path lengths. We tested our cohesion metric in annotated datasets and observed a high precision and recall in accurately identifying groups’ cohesions. Both methods were implemented on real Turtlebot 2 robots and tested with groups of pedestrians.

An important unexplored component of this research is the change in human behavior when they are approached by a robot in dense indoor environments. Robotics research is exploring friendlier modes of communicating and interacting with pedestrians. Our methods can complement works in human-robot interaction and enable robots to navigate social scenarios better.

Chapter 4: Navigating in Unstructured Outdoor Environments

4.1 Overview

The previous two chapters dealt with robots navigating in indoor settings with varying levels of crowd density. The discussed methods implicitly assume that the robot operates in smooth, flat ground. However, this assumption does not apply in unstructured outdoor settings, which constitute a distinct domain. Addressing the navigational challenges in outdoor environments is crucial, considering the wide range of applications such as food/grocery delivery, agriculture, surveillance, exploring forested and planetary environments, etc for which mobile robots are deployed in such settings.

The robots designed for these applications need to account for a terrain's geometric properties such as slope or elevation changes, surface characteristics, vegetation, etc to compute smooth and efficient robot trajectories [153–155].

Although a terrain's slope and elevation challenges are more readily detectable using range-based sensors like LiDARs, its surface properties (texture, bumpiness, and deformability), which also determine its navigability for a robot, are more subtle and can only be detected through RGB image data. Additionally, detecting and navigating vegetation with varying solidity properties (for example, tall grass being pliable while trees are not) that coexist in the same environment is extremely challenging. Such flora cannot be easily differentiated using range-based sensors

(e.g., laser scans, point clouds, and ultrasound) as they detect all plants as solid obstacles [156]. Furthermore, vegetation such as tall grass scatters laser beams from LiDARs and leads to poor characterization of their shape and structure [157]. In RGB images, the shape and structure of various plants are accurately represented. However, there is a lack of research on differentiating flora based on their pliability from images. Therefore, this dissertation focuses on the challenges of estimating the surface properties of a terrain, and the pliability of vegetation in unstructured outdoor environments using RGB images and using them for navigation.

A terrain's surface properties affect a (wheeled) robot's navigability on it in the following ways. For example, a surface's texture determines the traction experienced by the robot, its bumpiness determines the vibrations experienced, and deformability [158,159] determines whether a robot could get stuck or experience wheel slips (e.g. in mud). Other factors that affect navigability over a terrain involve the robot's properties such as dynamics, inertia, physical dimensions, velocity limits, etc.

The presence of dense vegetation on the other hand could lead to the robot not finding free space to circumvent the flora. This could cause the robot to freeze [54] similar to cases in dense crowds. Additionally, a small wheeled or legged robot could get physically entrapped in vegetation when its wheels or legs get intertwined in dried tall grass, bushes, etc. In such cases, the robot's dynamics determine whether it can autonomously recover itself [160]. To effectively navigate such environments, the robot must assess the traversability of the various flora around it. Firstly, it must differentiate flora based on their *pliability* [161]. We define a plant's pliability as its degree of flexibility or ability to bend such that a robot can navigate through it. For instance, robots can traverse *through* tall grass (high pliability) whereas trees are non-pliable and should be avoided. Secondly, the robot must detect the height and density of pliable vegetation since

they affect the resistance offered to the robot’s motion.

To address the first issue of terrain surface properties, many works in computer vision, specifically semantic segmentation [18,162,163] based on supervised learning, have demonstrated good terrain classification capabilities on RGB images. However, they rely on large image datasets with different classes of terrains annotated by humans. Such datasets do not account for a robot’s properties and might misclassify a traversable terrain for a robot as non-traversable (or vice-versa). In addition, their classification outputs must be converted into quantities that measure a terrain’s degree of navigability (costs) to be used for planning and navigation [28,164].

To overcome the aforementioned limitations, navigability costs could be directly predicted using input and label data collected in the real world through regression using *self-supervised learning* [158, 165]. That is, an image (input) can be associated with data vectors collected through other sensors such as Inertial Measurement Units (IMU) and wheel encoders on the robot (labels) instead of a human-provided label/annotation. Such label vectors generated from real-world sensor data could lead to a more accurate characterization of the robot-terrain interaction instead of a human-perceived ground truth.

Existing works that use self-supervised learning (online and offline) in the outdoor domain have predominantly focused on unstructured obstacle detection [155, 166], roadway and horizon detection [167], or long-range terrain classification into various categories [168–170]. Therefore, there is a lack of online self-supervised learning methods that can characterize a terrain’s surface properties. Many previous works also train their network offline by first collecting data before training on a powerful GPU [155, 158]. As a result, these methods can take a long time to re-train for a new kind of terrain.

To address the second issue of traversing dense vegetation, there have been several works

in computer vision to detect and segment simple vegetation such as trees, and short grass [171–173]. However, such works require extensive datasets with intricate annotations to train, requiring significant human effort. Additionally, segmenting traversable versus untraversable regions in dense vegetation is still an unsolved problem [174].

Navigation methods for outdoor domains have dealt with flora such as tall grass [156], trees, etc. in isolation. Such methods do not consider scenarios where plants of all pliability/traversability are in close proximity. Therefore, such methods cause the robot to collide or get entrapped, and do not have autonomous mechanisms to recover in such environments. Navigation methods based on end-to-end learning [156, 175] require real-world negative experiences such as collisions with non-pliable obstacles for training. This would be impractical for highly unstructured regions where rates of collision and entrapment are high.

To address these two major outdoor challenges, we propose the following two methods:

1. TerraPN (section 4.3, Fig. 4.1), a method that trains a neural network in an online self-supervised manner to learn a terrain’s surface properties (texture, bumpiness, deformability, etc.) and computes a robot-specific 2D surface cost map. TerraPN uses RGB images and the robot’s velocities as inputs, and processed IMU vibrations and a wheeled robot’s odometry errors as labels for training. It uses the computed cost map to guide a novel outdoor navigation algorithm that computes dynamically feasible collision-free trajectories by accounting for different surface costs. Our method adapts DWA’s [29] formulation by modulating the robot’s acceleration limits for different surfaces and computing trajectories with lower surface costs compared to DWA’s trajectories. Overall, TerraPN’s network trains in ~ 20 minutes for 5 different surfaces compared to segmentation methods that require 3-

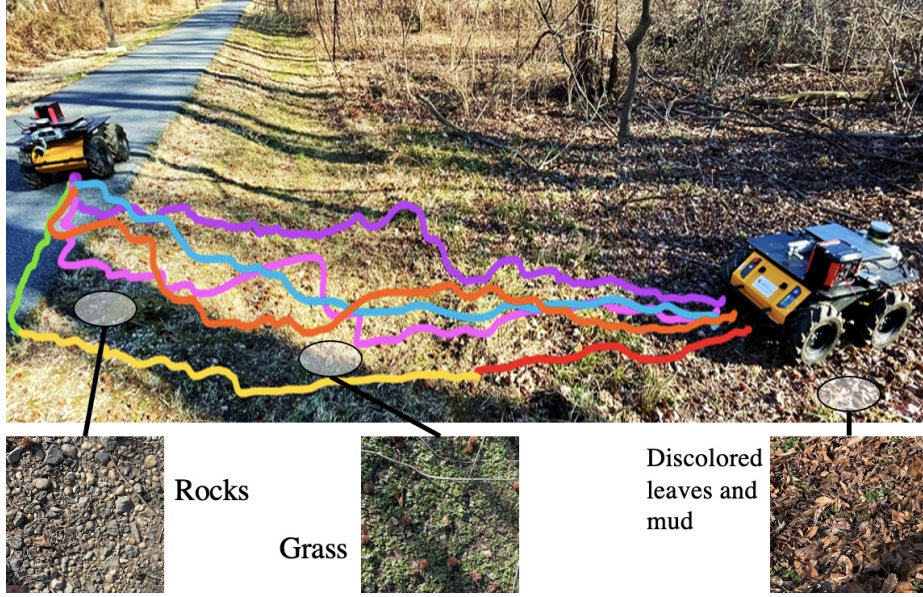


Figure 4.1: The trajectories of our method, TerraPN (green, yellow, red corresponding to fast, intermediate and slow speeds respectively), DWA [29] (blue), TERP [153] (orange), OCRNet-based (pink), and PSPNet-based (purple) navigation schemes in unstructured terrain. We observe that TerraPN’s trajectories navigate the robot on smooth surfaces (asphalt) as much as possible with high velocities and adapt the velocity to different surfaces based on their navigability costs. Other methods directly drive the robot towards its goal with its maximum velocity. In some cases (PSPNet-based), the trajectories are wandering since the segmentation falters when the surface becomes too unstructured.

10 hours of offline training to achieve similar levels of performances suitable for navigation.

Further, TerraPN results in improved navigation success rates (up to 35.84%), vibration costs (up to 21.52% lower), and mean velocity is high-cost surfaces (up to 46.76% lower) for different outdoor scenarios.

2. VERN (section 4.4, Fig. 4.2), a novel classifier to differentiate vegetation of various pliability/traversability from RGB images. VERN uses a novel few-shot learning approach and a Siamese network framework to train our classifier with only a few hundred images of different kinds of vegetation (tall grass, bushes/shrubs, and trees) with minimal human labeling. Furthermore, VERN computes a *vegetation-aware* traversability cost map by accounting for estimated vegetation height, pliability classification, and classification confidence.

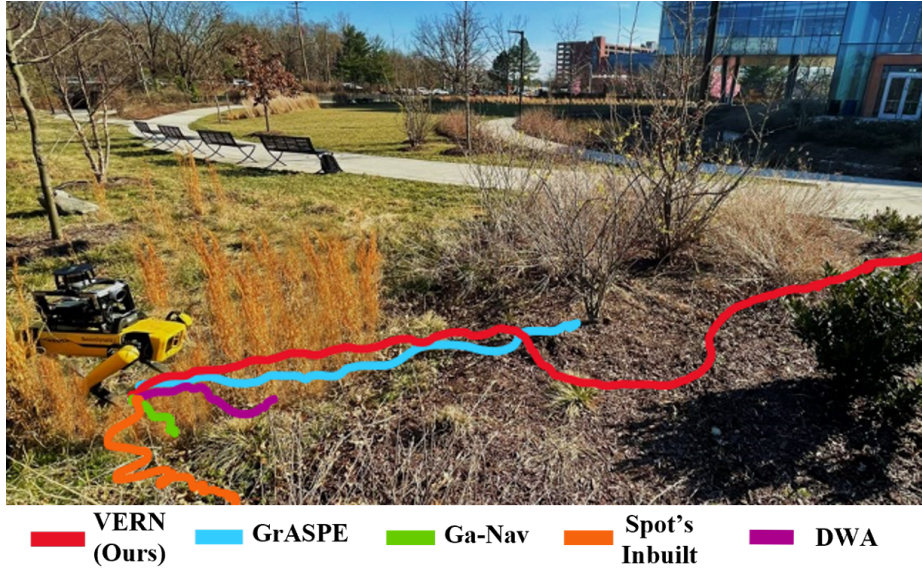


Figure 4.2: Comparison of VERN with other methods navigating a Spot robot through a complex environment (scenario 3) with traversable (e.g. tall grass), and untraversable (e.g. tree, bush) vegetation. In this trial, we observe that only VERN successfully reaches its goal due to its vegetation classification, height estimation, and novel cost map clearing scheme. Other methods either collide (GrASPE [176], Spot’s in-built autonomy), or freeze (DWA [177], GA-Nav [172]) due to the density of vegetation.

It also accounts for misclassifications in the vegetation classifier and leads to a more accurate representation of the traversable vegetation. Finally, VERN’s perception is integrated with a local planner that produces cautious navigation behaviors when in highly unstructured vegetation or regions with low confidence pliability classification. Additionally, the planner initiates novel holonomic behaviors to recover the robot if it freezes or gets physically entrapped in dense vegetation. This improves the success rate by up to 50%, and the freezing rate by 75%.

TerraPN and VERN have been implemented on a real Clearpath Husky robot, and a Boston Dynamics Spot robot, respectively. VERN was specifically designed for legged robots because wheeled robots lack the dynamics required to navigate through dense vegetation, and their wheels often get tangled in vegetation like tall grass. The performance of both methods has been evaluated and demonstrated in real-world unstructured outdoor environments (see Figs. 4.1, 4.7,

4.2, 4.11).

The rest of this chapter is organized as follows. In section 4.2, we discuss the related works in estimating terrain and vegetation traversability, and navigation in such scenarios. Sections 4.3 and 4.4 explain our two contributions in this domain. We finally summarize this chapter in section 4.5.

4.2 Related Work

In this section, we discuss previous works in computer vision for characterizing terrain and vegetation traversability and methods for navigating off-road terrains, and vegetation.

4.2.1 Characterizing Terrain Traversability

Traditional vision works have used methods such as Markov Random fields [178] and triangular mesh reconstruction [179] on 3D point clouds to analyze surface roughness and traversability. Learning-based works for traversability prediction fall into a combination of supervised/unsupervised and classification/regression categories.

Supervised Methods Works in pixel-wise semantic segmentation classify a terrain into multiple predefined classes such as traversable, non-traversable, forbidden, etc. [18, 162, 163]. To this end, fusing a terrain’s semantic and geometric features has also been studied [163]. These works are typically supervised and utilize large hand-labeled datasets of images [180, 181] to train classifiers. However, manually annotating datasets is time- and labor-intensive, not scalable to large amounts of data, and may not be applicable for robots of different sizes, inertias, and dynamics [182]. Such methods also assume that visually similar terrains have the same traversability

[166] without considering the robot’s dynamics, velocities, or other constraints.

Self-supervised Methods Unsupervised learning-based methods overcome the need for such datasets by automating the labeling process by either collecting terrain-interaction data such as forces/torques [158], contact vibrations [159], acoustic data [183], vertical acceleration experienced [184], and stereo depth [168, 169] and associating them with visual features (RGB data) for self-supervision. Other works have correlated 3D elevation maps and egocentric RGB images [28] or overhead RGB images [185] for classification.

Early self-supervised work [169, 186] focused on learning long-range ($\sim 100m$) terrain classification from stereo and RGB image data and 3D LiDAR points [170] and detecting roadways and the horizon using reverse optical flow [167] in a self-supervised manner. Many early works in self-supervised learning were proposed during the DARPA LAGR (Learning Applied to Ground Robots) program. For instance, [187] and [188] associated underfoot proprioception data with far-field exteroception data to estimate near to far-field terrain traversability. Works by Sermanet et al. [189] and Konolige et al. [190] expand further and integrate unsupervised learning-based terrain segmentation with mapping and localization for long-range navigation.

In another instance, Angelova et al. [191] train a non-linear regression model to correlate RGB images with wheel slip measurements to visually predict future wheel slips in granular/sandy surfaces. Subsequently, this was extended to using stereo images to first detect the terrain type, and then predicting slip based on its geometry [192]. Other works have correlated 3D elevation maps and egocentric RGB images [28] or overhead RGB images [185] for classification. However, none of these methods have accounted for the changes in terrain traversability based on the robot’s velocity.

Few methods have performed self-supervised regression, where for each RGB pixel in outdoor terrain, the corresponding force-torque measurements [158], trajectory model error [165], and resistive forces experienced [193] were predicted post-training.

4.2.2 Navigating Unstructured Terrains

Early works in outdoor navigation proposed using the binary classification of obstacles versus free space [194] and potential fields [41] for outdoor collision avoidance. With the advent of deep learning, methods to estimate navigability/energy cost in uneven terrains through imitation learning [154, 195] using egocentric sensor data and a priori environmental information [196] have been proposed. Siva et al. [197] addressed navigational setbacks due to wheel slip and reduced tire pressure in outdoor terrains by learning compensatory behaviors.

BADGR [155] presents an end-to-end DRL-based navigation policy that learns the correlation between events (collisions, bumpiness, and change in position) and the actions performed by the robot. Since learning-based approaches cannot guarantee any optimality in terms of a navigation metric (minimal cost, path length, dynamic feasibility, etc.), [153] proposed a hybrid model of spatial attention for perception and a dynamically feasible DRL method for navigation.

In TerraPN (section 4.3), we use a learning-based approach for the cost map prediction and a model-based (extending DWA [29]) approach for navigation. Therefore, TerraPN combines the benefits of accurate characterization of robot-terrain interaction and guaranteed low surface cost, and dynamically feasible navigation.

4.2.3 Perception in Dense Vegetation

Early works on vegetation detection were typically chlorophyll detectors [198–200], or basic classification models [201]. Nguyen et al. [199] proposed a sensor setup to detect near-IR reflectance and used it to define a novel vegetation index. [200] extended this detection by using an air compressor device to create strong winds and estimating the levels of resistance to robot motion using the movement of vegetation. However, it needed the robot to be static. Multiple sensor configurations have also been explored for obstacle detection within vegetation [202] of which thermal camera and RADAR stand out as effective modalities.

Modeling the frictional/lumped-drag characteristics of vegetation [161,203,204] as a measure of the resistance offered to motion, or modeling plant stems using their rotational stiffness and damping characteristics [205] has also been proposed. However, the method does not use visual feedback, requiring the robot to drive through vegetation first to gauge its pliability.

Wurm et al. [206,207] presented methods to detect short grass from LiDAR scans based on its reflective properties on near IR light. Astolfi et al. [208] demonstrated accurate SLAM and navigation through sensor fusion in a vineyard. However, such methods operated in highly structured environments.

There are several works that utilize semantic segmentation to understand terrain traversability [172, 173, 209]. [209] applied semantic segmentation classifier trained on RGB images to oct-tree maps obtained from RGB-D point cloud. Maturana et al. [173] took a similar approach by training a segmenter and augmenting a 2.5D grid map with it to distinguish tall grass from regular obstacles. However, these existing methods are not suitable for perception in dense, unstructured vegetation.

4.2.4 Navigating Dense Vegetation

There have been very few methods for detecting and navigating through vegetation. Overbye et al. [210] proposed using a terrain gradient map along with the A* algorithm to navigate a large wheeled robot and demonstrated moving over tall grass and short bushes. [211] extended this by adding a segmentation layer to the map to detect soft obstacles. However, these methods mostly operate in structured, isolated vegetation and may not work well in unstructured scenarios.

There are several works that have addressed navigating through pliable vegetation such as tall grass [156, 175]. Kahn et al. [156] demonstrated a model that learns from a robot’s real-world experiences such as collisions, bumpiness, and its position to navigate outdoor environments. The robot learned from RGB images and associated experiences (labels) to consider tall grass as traversable. However, it does not account for the presence of non-traversable bushes or trees alongside traversable vegetation. Polevoy et al. [175] proposed a model that regresses the difference between the robot’s dynamics model and its actual realized trajectory in unstructured vegetation. This acts as a measure of the surrounding vegetation and terrain’s traversability. However, both these methods require “negative” examples such as collisions during training, which may be impractical or dangerous to collect in the real world.

With the advent of legged robots, several recent works have been focused on developing robust controllers for locomotion purely using proprioception [212] or fusing it with exteroceptive perception such as elevation maps [213]. A few navigation works focus on estimating the underlying support surface that is hidden by vegetation [214] by fusing haptic feedback from the robot, depth images, and detecting the height of the vegetation. VERN 4.4 complements existing works by enhancing a legged robot’s ability to navigate through dense vegetation and

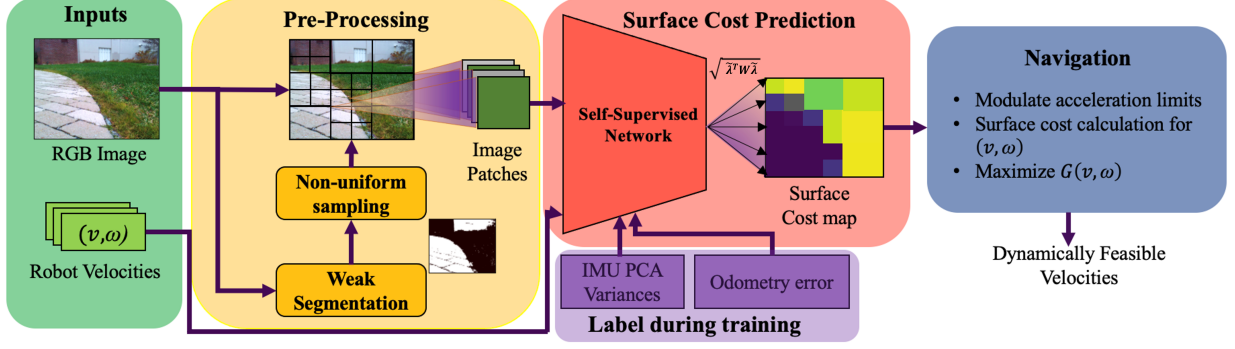


Figure 4.3: TerraPN’s overall architecture. Our method uses RGB images and a set of robot velocities to predict a surface cost map for the robot. In the pre-processing step, a classical image segmentation method (Weak segmentation) is used to differentiate regions with different pixel intensities. A non-uniform patch sampling scheme is then applied where large/small patches are extracted from regions that have the same/varied pixel intensities. The patches are passed into our self-supervised network (see Fig. 4.9) that predict a 4×1 label vector $\tilde{\lambda}$. We compute the surface cost as $\tilde{\lambda}$ ’s weighted norm, create patches with the cost values, and concatenate them based on their locations on the RGB image to form the cost map (blue implies low, yellow implies high costs). The cost map is then used by our novel navigation algorithm to compute dynamically feasible velocities with low surface costs in real-time. This velocity computation task is performed iteratively at every time step for a given image and velocity vector input pair.

providing a method for recovery when it becomes entrapped.

4.3 TerraPN

In this section, we elaborate on TerraPN’s methodology (sections 4.3.2 and 4.3.3). But first, we explain some of the preliminary concepts used in its formulation in section 4.3.1. Finally, we elaborate the details of its implementation and evaluations, and its limitations in sections 4.3.4 and 4.3.5, respectively.

4.3.1 Preliminaries

In this section, we define TerraPN’s problem formulation and provide some background on DWA [29] that forms the basis for our outdoor navigation.

4.3.1.1 Problem Formulation

The problem of navigating in outdoor environments with various surface properties can be divided into two stages: surface cost prediction, and low-cost navigation. To predict surface costs, we train a neural network that uses RGB image patches $I_{RGB}^{n \times n}$, a set of the robot’s linear and angular velocities (v, ω) as inputs, and a 4×1 label vector λ consisting of two dimensions corresponding to IMU measurements, and two to robot’s odometry errors. The IMU measurements and odometry errors characterize a surface’s traction, bumpiness, and deformability (see section 4.3.2.2). After training, we use the neural network’s predicted label $\tilde{\lambda}$ to construct a 2D surface cost map. The cost map is a non-uniformly discretized concatenation of patches that depend on the surface boundaries in the input image.

Next, the cost map is used by the navigation component to compute dynamically feasible trajectories with low surface costs. We use i, j for denoting various indices, x, y to denote positions relative to different coordinate frames. Our method’s overall architecture is shown in 4.3.

4.3.1.2 Dynamic Window Approach

DWA is a model-based collision avoidance algorithm that guarantees dynamically feasible robot velocities. However, its formulation implicitly assumes that the robot traverses on a uniform, smooth navigable surface. TerraPN’s navigation component modifies and extends DWA’s formulation for outdoor navigation by accounting for surface costs.

DWA’s formulation involves two stages: 1. computing a collision-free, dynamically feasible velocity search space, and 2. choosing a velocity in the search space to maximize an objective

function.

In the first stage, all possible velocities in $V_s = [v \in [0, v_{max}], \omega \in [-\omega_{max}, \omega_{max}]]$, respectively, are considered for the search space V_s . Here, v_{max}, ω_{max} represent the robot's maximum achievable linear and angular velocities respectively. Next, all the (v, ω) pairs that prevent a collision in V_s are used to form the admissible velocity set V_a . Lastly, the velocity pairs that are reachable, accounting for the robot's acceleration limits $\dot{v}_{max}, \dot{\omega}_{max}$ within a short time interval Δt , are considered to construct a dynamic window set V_d . The resulting search space is constructed as $V_r = V_s \cap V_a \cap V_d$.

In the second stage, DWA searches for $(v, \omega) \in V_r$, which maximizes the following objective function.

$$G_1(v, \omega) = \alpha.head + \beta.dist + \gamma.vel \quad (4.1)$$

Here, $head()$ measures the progress towards the robot's goal, $dist()$ is the distance to the closest obstacle when executing a certain (v, ω) , and $vel()$ measures the forward velocity of the robot and encourages higher velocities. v, ω on the RHS are omitted for clarity.

4.3.2 TerraPN: Predicting Surface Costs

In this section, we first describe the components in computing a terrain's surface cost map:

1. data collection, 2. network architecture and training, 3. cost map generation using variable sampling.

4.3.2.1 Autonomous Data Collection

To generate the input and label data for training the cost map prediction network (Fig. 4.9), we collect the raw sensor data from an RGB camera, robot’s odometry, 6-DOF IMU, and 3D LiDAR autonomously on different surfaces. The robot performs a set of maneuvers in two different speed ranges: 1. slow ($[0, \frac{v_{max}}{2}]$ m/s and $[-\frac{\omega_{max}}{2}, \frac{\omega_{max}}{2}]$ rad/s), and 2. fast ($[0, v_{max}]$ m/s and $[-\omega_{max}, \omega_{max}]$ rad/s). The maneuvers include: 1. moving along a rectangular path, 2. moving in a serpentine trajectory, and 3. random motion.

The maneuvers are designed to cover all the $[v, \omega]$ pairs within the robot’s velocity limits to emulate different terrain interactions while the data is collected. If the robot encounters an obstacle, it switches from executing the maneuvers to avoiding a collision using DWA [29].

4.3.2.2 Computing Inputs and Labels

As mentioned in section 4.3.1.1, our network uses an $n \times n$ patch ($I_{RGB}^{n \times n}$) that is cropped from the center bottom of the collected full-sized image of size $w \times h$ ($I_{RGB}^{w \times h}$). To generate the velocity input, the linear and angular velocities for the past $n/2$ instances from the robot’s odometry are obtained and reshaped to a $2 \times n/2$ vector. The velocity vector’s dimensions are chosen such that the image input does not dominate the network’s predictions.

Our label vector consists of IMU and odometry error components. They are robot-specific and implicitly encode the robot’s dynamics, inertia, etc., and its interactions with the terrain. To generate the IMU component, we apply Principal Component Analysis (PCA) to reduce the dimensions of the collected 6-dimensional IMU data (linear accelerations and angular velocities) to its two principal components. From Fig. 4.4, we observe that the variances (σ_{PC1} and σ_{PC2})

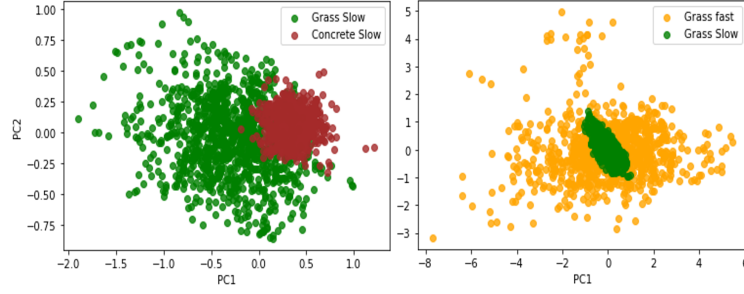


Figure 4.4: Sample results of the PCA applied on the 6-dimensional IMU data. The variances of the data along the two principal axes for **[Left]** two different surfaces concrete (red), and grass (green), and **[Right]** two different velocity ranges for the same surface. We observe that the variances can differentiate various surfaces and speed levels.

of the data along the principal components help differentiate various surfaces in terms of their bumpiness. Additionally, for the same surface, higher velocities lead to higher variances in the data (justifying the need to consider velocities as inputs).

To generate the odometry error component of the label vector, the distance traveled by the robot (Δd_{odom}) and its change in orientation ($\Delta \theta_{odom}$) in a time interval Δt are obtained from the robot's odometry. We obtain the same data (Δd_{loam} , $\Delta \theta_{loam}$) from a 3D LiDAR-based odometry and mapping system [215]. The distance and orientation change errors are calculated as,

$$d_{error} = \Delta d_{loam} - \Delta d_{odom} \quad (4.2)$$

$$\theta_{error} = \Delta \theta_{loam} - \Delta \theta_{odom}. \quad (4.3)$$

This component of the label vector differentiates surfaces with high deformability or poor traction where, if the robot's wheels get stuck or slip (e.g. in mud), $\Delta d_{loam} \approx 0$ and $\Delta \theta_{loam} \approx 0$, whereas Δd_{odom} and $\Delta \theta_{odom}$ would have high values. The final label vector is given by,

$$\lambda = [\sigma_{PC1} \quad \sigma_{PC2} \quad d_{error} \quad \theta_{error}]^T. \quad (4.4)$$

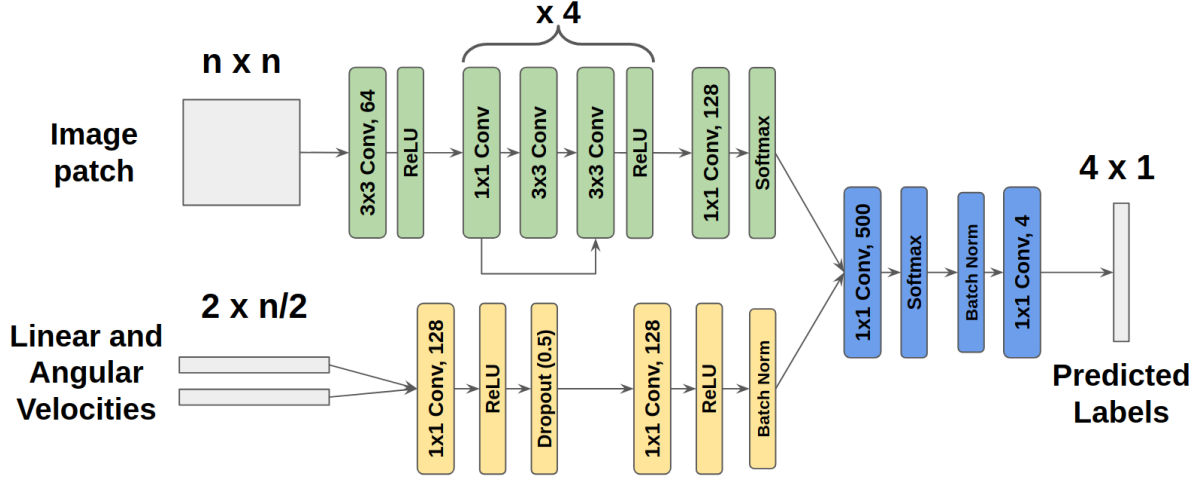


Figure 4.5: Our novel two-stream network architecture. One stream encodes image patches (green) and the other stream processes the linear and angular velocities (yellow). The feature embeddings from each stream are concatenated and finally passed into a set of linear layers (blue), and the final predictions are the estimated IMU readings and the odometry errors associated with this image patch.

4.3.2.3 Network Architecture and Online Training

The network (see Fig. 4.9) is trained to predict the 4×1 vector in equation 4.4 given the image and velocity inputs. The architecture uses a series of 2D convolution with skip connections and batch normalization on the image input, and several layers of fully connected layers with dropout and batch normalization for the velocity input, shown in Fig. 4.9. In the image stream, after the initial convolution operation with ReLU activation, the image is connected to four residual blocks and one linear layer with batch normalization. Since we have limited data when collecting labels and training the network online, we use the residual connection to ensure that the network would not overfit the collected data given many layers and parameters in the network. Dropout and batch normalization layers also improve generalization capabilities and avoid overfitting.

As the robot autonomously collects sensor data on different surfaces, the inputs and labels

are generated and shuffled. The network's training is started and performed online once sufficiently varied data is collected (~ 6 minutes). The data collection and online training is normally completed in 20 – 25 minutes.

4.3.2.4 Navigability Cost Calculation

Based on the vector predicted by the network ($\tilde{\lambda} = [\tilde{\sigma}_{PC1} \quad \tilde{\sigma}_{PC2} \quad \tilde{d}_{error} \quad \tilde{\theta}_{error}]^\top$), the navigability cost for a given RGB patch and velocity vector is computed as the weighted norm of $\tilde{\lambda}$,

$$c = \sqrt{\tilde{\lambda}^\top \mathbf{W} \tilde{\lambda}} \quad (4.5)$$

Here, $\mathbf{W}$ is a diagonal matrix with positive weights. To make navigability cost predictions on a full-sized RGB image $I_{RGB}^{w \times h}$, the image is first resized into new dimensions as follows,

$$w' = \lfloor w/n \rfloor \cdot n, \quad h' = \lfloor h/n \rfloor \cdot n, \quad (4.6)$$

where $\lfloor \cdot \rfloor$ is the nearest integer operator. Next, non-overlapping $n \times n$ patches are cropped along the width and height of the image. This $\lfloor w/n \rfloor \cdot \lfloor h/n \rfloor$ batch of images is passed as inputs along with a batch containing the input velocity vectors to obtain the navigability cost predictions c_{ij} ($i \in [1, \lfloor w/n \rfloor]$ and $j \in [1, \lfloor h/n \rfloor]$) for different regions of the resized image $I_{RGB}^{w' \times h'}$. The predicted costs are normalized to be in the range $[0, \pi/2]$.

Finally, the surface navigability cost map $C^{w' \times h'}$ is constructed by vertically and horizontally concatenating $n \times n$ patches with the values of c_{ij} corresponding to different regions. $C^{w' \times h'}$ is then resized back to $C^{w \times h}$.

4.3.2.5 Non-uniform Patch Sampling

Although using patches reduces the input dimensionality of our network and helps train it faster, it could result in regions with multiple surfaces (boundaries) in the full-sized image having inaccurate costs depending on the patch size. Therefore, we employ a non-uniform patch sampling technique to obtain finer patches in multi-surface regions (boundaries). Conversely, portions with a single surface can be sampled with larger patch sizes $2n, 4n$. This reduces the total number of patches used for cost prediction on the full-sized image, thus maintaining the method's inference rate.

Weak Segmentation: To detect regions with multiple surfaces and differentiate them in a scene, we use a *weak* segmentor which is based on classical image segmentation methods. The weak segmentation method is computationally light (inference time of $\sim 66\mu s$) and may not have pixel-level precision. But, it sufficiently demarcates the boundaries of various terrains in the input image. First, the Sobel edge detector is applied to the grayscale input image $I_{gray}^{w' \times h'}$ of the scene, and the histogram of the result is computed. Next, based on the Bayesian information criterion [216] a Gaussian mixture model is fit to the histogram, and the mean of each Gaussian curve is used as a marker/threshold $(\mu_1, \mu_2, \dots, \mu_k)$ to differentiate the regions of pixels with different intensity levels in the image. Finally, the watershed filter [217] is applied to highlight the regions of different intensities to obtain $I_{WS}^{w \times h}$ (See figure 4.3).

Selecting Sampling Patch Size: We consider the patches $I_{WS}^{n \times n}, I_{WS}^{2n \times 2n}, I_{WS}^{4n \times 4n}$ in the weak segmentation output $I_{WS}^{w' \times h'}$ and the intensity of pixels within them. If a patch larger than $n \times n$ satisfies the condition in 4.7, smaller patches are not considered for cost prediction.

$$\frac{num(\mu_i)}{n^2} > \xi \quad i \in [1, k]. \quad (4.7)$$

where $num(\mu_i)$ is the number of pixels with intensity greater than or equal to μ_i , and ξ is a threshold. This condition ensures that when a large patch has a significant number of pixels with the same intensity, implying the presence of a single surface, smaller patches are not used for cost prediction. The larger patch is resized to $n \times n$ before passing into the network.

4.3.3 TerraPN: Navigation

In this section, we explain how the computed surface cost map is used by TerraPN's navigation to adapt DWA (section 4.3.1.2) for outdoor navigation and compute dynamically feasible, low surface cost velocities.

4.3.3.1 Trajectory Navigability Cost

To adapt DWA to outdoor terrains, the trajectory corresponding to a (v, ω) pair must be associated with a surface cost. The trajectory for a given (v, ω) pair relative to a coordinate frame attached to the robot's center (X-axis pointing forward, Y-axis pointing left) is calculated as,

$$\begin{aligned} x_i^{rob} &= v \cos(\omega t_i) t_i \\ y_i^{rob} &= v \sin(\omega t_i) t_i \\ t_i &= t_0 + i \Delta t, \quad i \in [0, s_{num}]. \end{aligned} \quad (4.8)$$

Here, t_0 is the initial time instant and s_{num} is the number of time steps used for extrapolating the trajectory. This trajectory is then transformed relative to the camera frame attached to the robot using a homogeneous transformation matrix as $[x_i^{cam} \ y_i^{cam}]^\top = H_{rob}^{cam} \cdot [x_i^{rob} \ y_i^{rob}]^\top$. Next,

the trajectory is converted to correspond to the image/pixel coordinates of $I_{RGB}^{w' \times h'}$, i.e., $[x_i^{img} \ y_i^{img}]^\top$ using the camera's intrinsic parameters. The navigability cost for a velocity pair can be then computed from $C^{w' \times h'}$ as,

$$sur(v, \omega) = \sum_{i=0}^{s_{num}} cost(x_i^{img}, y_i^{img}). \quad (4.9)$$

Here, $cost()$ is the surface cost at a given pixel's coordinates.

4.3.3.2 Variable Acceleration Limits

Robot navigation methods consider a constant range of linear $([-\dot{v}_{max}, \dot{v}_{max}])$ and angular $([-\dot{\omega}_{max}, \dot{\omega}_{max}])$ accelerations. Our formulation varies the linear and angular acceleration limits available for planning depending on the properties of the surface on which the robot is traversing such that $\dot{v} \in [-\dot{v}_{max}, \dot{v}_{lim}]$ and $\dot{\omega} \in [-\dot{\omega}_{lim}, \dot{\omega}_{lim}]$.

This is done because, intuitively, the robot accelerating on a smooth surface (e.g., concrete, asphalt) would lead to a low navigability cost. Therefore, the robot can proceed towards its goal faster. Whereas on a bumpy surface or one with poor traction, (e.g., tiled surface, dry leaves), accelerating would lead to high vibrations and the risk of getting stuck (e.g. mud). We do not limit the maximum deceleration available to the robot since it may have to slow down to avoid obstacles or while moving on a rough surface.

First, we divide the trajectory corresponding to the robot's current (v, ω) and calculate the cost for the second half as follows,

$$\mathcal{C}_{\frac{s_{num}}{2}+1:s_{num}} = \sum_{i=\frac{s_{num}}{2}+1}^{s_{num}} cost(x_i^{img}, y_i^{img}) / (s_{num}/2). \quad (4.10)$$

We limit the robot's accelerations using this navigability cost as,

$$\dot{v}_{lim} = \tau \cdot \dot{v}_{max}, \quad (4.11)$$

$$\dot{\omega}_{lim} = \tau \cdot \dot{\omega}_{max}, \quad (4.12)$$

$$\tau = \cos(\mathcal{C}_{\frac{s_{num}}{2}:s_{num}}), \quad \mathcal{C}_{\frac{s_{num}}{2}:s_{num}} \in [0, \pi/2]. \quad (4.13)$$

If $\mathcal{C}_{\frac{s_{num}}{2}+1:s_{num}}$ is low (low-cost surface), the robot is allowed to accelerate towards its goal, while a high $\mathcal{C}_{\frac{s_{num}}{2}+1:s_{num}}$ restricts the robot from speeding up. Considering only the second half of the trajectory also implicitly accounts for transitions between surfaces. Using these acceleration limits, a new dynamic window V'_d is constructed. The new resultant search space is calculated as $V'_r = V_s \cap V_a \cap V'_d$.

4.3.3.3 Optimization

Finally, a $(v, \omega) \in V'_r$ which maximizes the objective function

$$G_2(v, \omega) = \alpha.head + \beta.dist + \gamma.vel - \delta.sur, \quad (4.14)$$

is chosen. Here, $\alpha, \beta, \gamma, \delta$ are weights for each component.

Lemma 4.3.1. *TerraPN's navigation computes collision-free, dynamically feasible trajectories with surface costs that are always lesser than or equal to the DWA's trajectory's surface cost.*

Proof. Let $(v_1, \omega_1) = \operatorname{argmax}(G_1)$, and $(v_2, \omega_2) = \operatorname{argmax}(G_2)$, $\forall v, \omega \in V'_r$. From equations 4.1 and 4.14, we get $G_2(v, \omega) = G_1(v, \omega) - \delta.sur(v, \omega)$. We know that, $G_2(v_2, \omega_2) \geq G_2(v_1, \omega_1) \implies G_1(v_2, \omega_2) - \delta.sur(v_2, \omega_2) \geq G_1(v_1, \omega_1) - \delta.sur(v_1, \omega_1)$. Rearranging the terms, we get $G_1(v_1, \omega_1) - G_1(v_2, \omega_2) \leq \delta(sur(v_1, \omega_1) - sur(v_2, \omega_2))$.

Since v_1, ω_1 maximizes G_1 , in the LHS, $G_1(v_1, \omega_1) - G_1(v_2, \omega_2) \geq 0 \implies sur(v_1, \omega_1) - sur(v_2, \omega_2) \geq 0$. This inequality holds since $sur(v, \omega) \geq 0$. The dynamical feasibility of TerraPN's velocities follows from the fact that acceleration limits obey $[-\dot{v}_{max}, \dot{v}_{lim}] \subseteq [-\dot{v}_{max}, \dot{v}_{max}]$ as $\tau \in [0, 1]$. ■

4.3.3.4 Algorithm

The algorithm for TerraPN's outdoor navigation is shown in Algorithm 2.

Algorithm 2: TerraPN Navigation

```

1  Input obs, goal,  $\dot{v}_{max}, \dot{\omega}_{max}, v_{curr}, \omega_{curr}, s_{num}$ 
2  Output  $v^*, \omega^*$ 
   1:  $V_s = [[0, v_{max}], [-\omega_{max}, \omega_{max}]]$ 
   2:  $V_A = \text{CollisionFree}(V_s, obs)$ 
   3:  $\text{traj}_{curr} = \text{ComputeTrajectory}(v_{curr}, \omega_{curr}, s_{num})$ 
   4:  $C_{\frac{s_{num}}{2}+1:s_{num}} = \text{SurfaceCost}(\text{traj}_{curr}, s_{num})$ 
   5:  $\tau = \cos(C_{\frac{s_{num}}{2}+1:s_{num}})$ 
   6:  $\dot{v}_{lim}, \dot{\omega}_{lim} = \tau \cdot \dot{v}_{max}, \tau \cdot \dot{\omega}_{max}$ 
   7:  $V'_d = [[v_{curr} - \dot{v}_{max}\Delta t, v_{curr} + \dot{v}_{lim}\Delta t], [\omega_{curr} - \dot{\omega}_{lim}\Delta t, \omega_{curr} + \dot{\omega}_{lim}\Delta t]]$ 
   8:  $V'_r = V_s \cap V_A \cap V'_d$ 
   9:  $G_2 = \alpha \cdot head(V'_r, goal) + \beta \cdot dist(V'_r, obs) + \gamma \cdot vel(V'_r) - \delta \cdot sur(V'_r)$ 
  10:  $v^*, \omega^* = \underset{v, \omega \in V'_r}{\text{argmax}}(G_2)$ 

```

Here, obs is the set of obstacles in the environment characterized by LiDAR scans. In line 2, CollisionFree() is a function that returns the velocities that avoid a collision with the obstacles in the environment. The ComputeTrajectory() function in line 3 implements equation 4.8, and the SurfaceCost() in line 4 implements equation 4.10.

4.3.4 Results and Analysis

We detail our method’s implementation, our evaluation metrics, and the different environments we tested in, and compare with other methods in this section.

4.3.4.1 Implementation

Our self-supervised learning network is implemented using Tensorflow. A Clearpath Husky robot mounted with a VLP-16 LiDAR and Intel Realsense camera is used to evaluate our method in real-world scenarios. The LiDAR is only used for the initial LiDAR-based odometry data collection and to provide 2D scans for detecting obstacles for navigation. For processing, the robot is equipped with a laptop with an Intel i9 processor and Nvidia RTX2080 GPU.

In our formulation, we use $n = 50$, $w = 640$, $h = 480$, $\xi = 0.8$, $s_{num} = 15$, $\Delta t = 0.1$, $\alpha = 2.4$, $\beta = 3.2$, $\gamma = 0.1$, $\delta = 50$. The velocity and acceleration limits are set according to the Husky’s limits. We train our cost map prediction on 5 surfaces: 1. concrete, 2. tiles, 3. grass, 4. asphalt, and 5. fallen yellow leaves. Each surface offers different levels of traction, bumpiness, and deformability.

4.3.4.2 Navigation Evaluation Metrics

We use the following metrics for our comparisons.

Success Rate - The number of times the robot reached its goal while avoiding getting stuck or colliding over the total number of trials.

Normalized Trajectory length - The robot’s trajectory length normalized over the straight-line distance to the goal for all the successful trials.

Vibration Cost - The summation of the vertical motion's gradient experienced by the robot along its trajectory.

Mean Velocity - The robot's average velocity along its trajectory as it traverses various surfaces.

We compare our method with DWA [29], and TERP [153] which navigates the robot based on the elevation maps in the environment. We also compare with two methods that use OCRNet [218], and PSPNet [219] for semantic segmentation and the Dijkstra's algorithm for waypoint computation and navigation. OCRNet and PSPNet classify different surfaces into several discrete classes based on their degree of navigability. Different cost values are associated with each class and used for waypoint computation using Dijkstra's algorithm. The OCRNet and PSPNet networks were trained on the RELIS-3D outdoor dataset which contains all the surfaces TerraPN's cost prediction network is trained for.

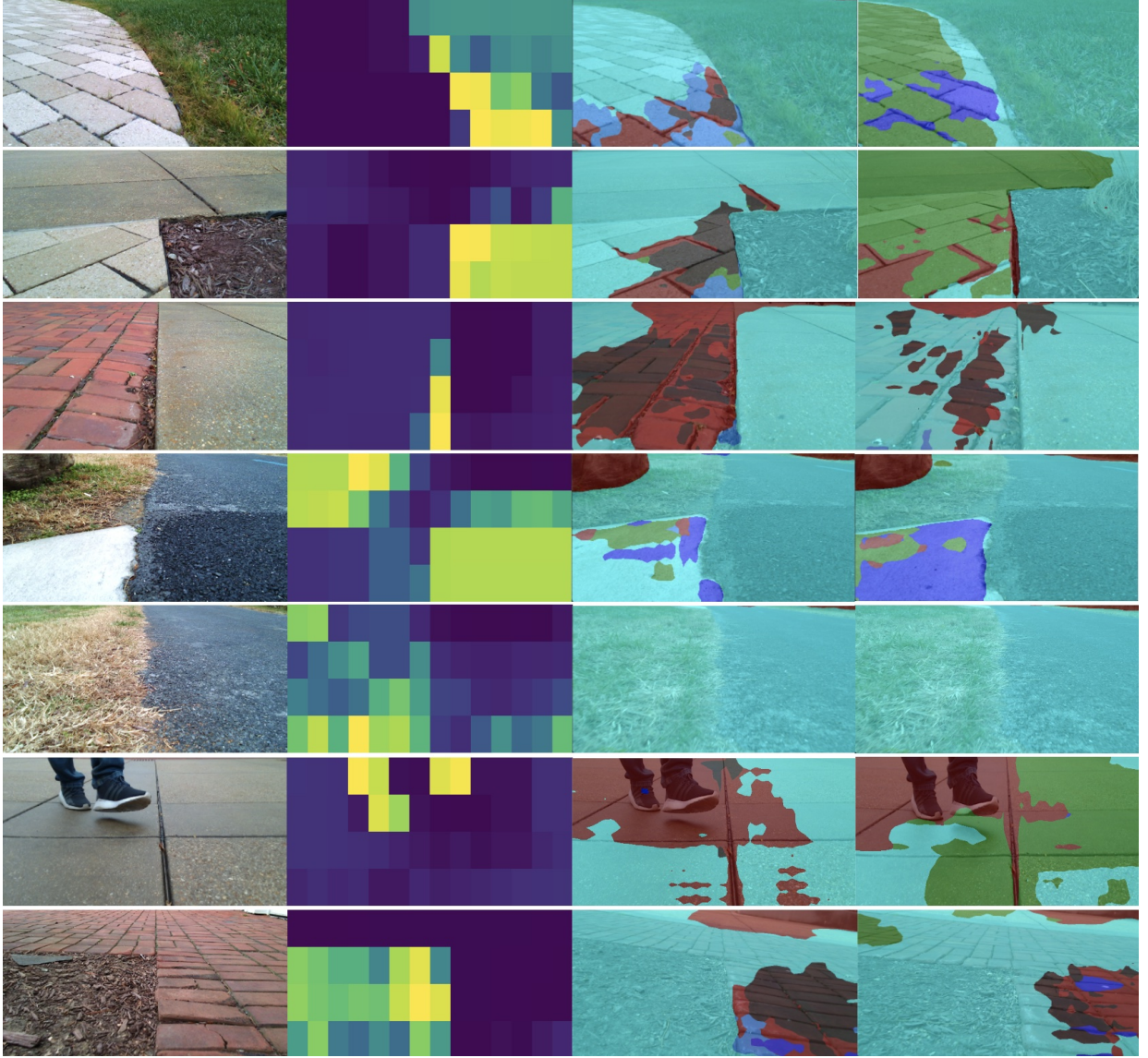


Figure 4.6: Surface cost map predictions along with OCRNet’s (third column) and PSPNet’s (fourth column) segmentation results for different scenarios with 2-3 surfaces. TerraPN’s cost prediction considers non-uniformly sampled patches on the RGB images and feeds them into our network along with the robot’s velocities. We observe that surfaces are differentiated well based on their navigability (dark blue denotes low cost and therefore better navigable surface and lighter, yellow shades denote high cost). We observe that our cost predictions are fairly accurate even for regions not seen during training (person’s legs, soil, boundaries between surfaces). OCRNet and PSPNet mostly classify surfaces as traversable (green), or moderately traversable (cyan). However, they often incorrectly segment some patches as forbidden (red) or as obstacles (blue) which reflect poorly during navigation.

4.3.4.3 Testing Scenarios

We evaluate and compare TerraPN’s navigation with prior methods in five scenarios. We characterize each scenario’s difficulty based on the number of surfaces, the level of unstructuredness, and whether they were previously seen during TerraPN’s cost prediction training.

- **Scenario 1:** Two trained surfaces (concrete and grass). See Fig. 4.7a.
- **Scenario 2:** Three surfaces (concrete, asphalt, and rocks) with one untrained surface (rocks). See Fig. 4.7b.
- **Scenario 3:** Four surfaces (tiles, concrete, mud, grass) with one untrained surface (mud) where the robot could get stuck. See Fig. 4.7c.
- **Scenario 4:** Four surfaces (asphalt, rocks, discolored grass, unstructured dry brown leaves with mud). The grass and dry leaves surfaces had undergone considerable seasonal changes. See Fig. 4.1.
- **Scenario 5:** Untrained, highly unstructured surface with dry brown leaves, broken branches, mud, etc. See Fig. 4.7d.

4.3.4.4 Analysis and Comparisons

Cost Map Prediction Performance: Fig. 4.6 shows the results of our cost map prediction in scenarios with various surfaces, both seen during training (tiles, grass) and unseen (soil, rocks, human obstacles). For unseen surfaces, our network typically predicts higher costs for

Metrics	Method	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Success Rate (%) (Higher is better)	DWA [29]	100	70	79	53
	TERP [153]	100	77	78	61
	OCRNet [218]	94	72	73	58
	PSPNet [219]	92	75	72	56
	TerraPN	100	88	85	72
Norm. Traj. Length (Close to 1 is better)	DWA [29]	0.965	1.001	1.076	1.141
	TERP [153]	0.996	1.271	1.158	1.229
	OCRNet [218]	1.389	1.084	1.403	1.287
	PSPNet [219]	1.241	1.066	1.524	1.369
	TerraPN	1.147	1.034	1.127	1.261
Vibration Cost (lower is better)	DWA [29]	2.334	1.678	1.518	3.652
	TERP [153]	1.199	1.279	1.627	4.156
	OCRNet [218]	0.893	2.115	1.393	4.378
	PSPNet [219]	0.967	2.384	1.424	4.456
	TerraPN	0.766	1.329	1.166	2.886
Mean Velocity (lower is better)	DWA [29]	0.581	0.564	0.531	0.542
	TERP [153]	0.544	0.506	0.525	0.522
	OCRNet [218]	0.561	0.532	0.529	0.515
	PSPNet [219]	0.548	0.526	0.509	0.513
	TerraPN	0.347	0.336	0.271	0.296

Table 4.1: Relative performance of our method TerraPN compared to other methods on various metrics. We observe that TerraPN leads to the highest success rates in all the scenarios. Further, TerraPN results in vibration cost decrease up to 21.52%, mean velocity reduction up to 46.76%, and shorter or comparable trajectory lengths than the other segmentation methods in most of the scenarios. DWA and TERP take shorter trajectories in some cases since they directly move toward the goal in the absence of obstacles and elevation changes, without considering surface properties.

Method	Inference Time (sec)	Training Time
OCRNet	0.052	10hrs and 5mins
PSPNet	0.045	2hrs and 47mins
CGNet	0.015	9hrs and 53mins
TerraPN-50	0.055	20-25mins
TerraPN-non-uniform	0.029	20-25mins

Table 4.2: TerraPN’s inference time and training time compared to existing semantic segmentation methods are executed on a laptop with Nvidia RTX2080 GPU. We observe that TerraPN’s non-uniform sampling approach reduces the inference time significantly compared to PSPNet, OCRNet, and uniform sampling of 50×50 patches (TerraPN-50). Even though CGNet outperforms all the other methods in terms of inference time, it needs ~ 10 hours of training time to achieve satisfactory segmentation accuracy. However, TerraPN achieves cost prediction results suitable for outdoor navigation within ~ 25 minutes of training for 5 different surfaces.

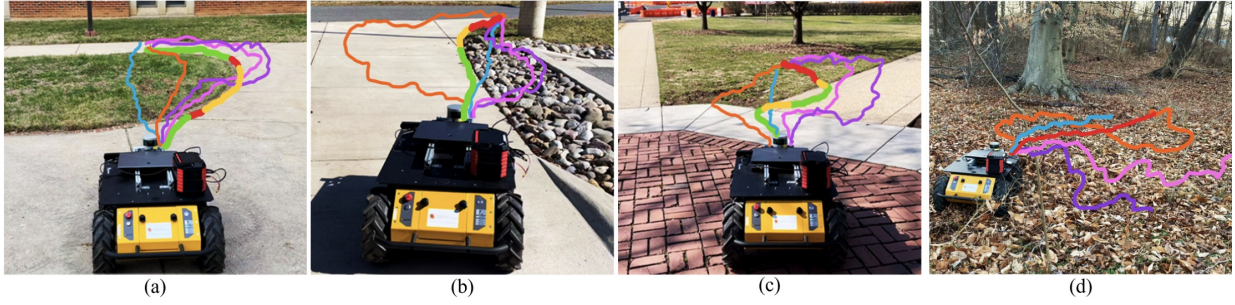


Figure 4.7: Robot trajectories when navigating in different unstructured terrains using our method, TerraPN (green, yellow, red corresponding to different speed levels), DWA [29] (blue), TERP [153] (orange), OCRNet-based (pink), PSPNet-based (purple) navigation schemes. (a) Scenario 1; (b) Scenario 2; (c) Scenario 3; (d) Scenario 5. We observe that TerraPN generates relatively shorter or comparable trajectories that navigate the robot on low-cost surfaces (concrete, asphalt, etc). Further, TerraPN varies the velocity appropriately when the robot encounters a relatively rough surface such as grass, rocks, dry leaves, or mud, while other methods always navigate the robot at its maximum speed on all surfaces. This leads to high vibrations and odometry errors in the robot. In (d), TerraPN moves slowly (red) throughout its progress towards its goal. TERP takes a much longer trajectory to the goal based on elevation changes. All other methods fail to reach the goal either due to incorrect segmentation (OCRNet, PSPNet) or high odometry errors (DWA).

navigation, which results in cautious, slow trajectories near such surfaces. We observe a clear differentiation of the surfaces (blue denoting low-cost surfaces) based on the robot’s current velocity. Additionally, non-uniform patch sampling uses smaller patches in certain regions of interest such as the boundaries between different surfaces, and even over a pedestrian’s shoes. In all other regions, larger patches are used, achieving a good balance between the fineness of the cost map and computation time.

OCRNet and PSPNet mostly segment different surfaces as traversable (green) and moderately traversable (cyan). However, in many instances small regions of forbidden (red) and obstacle (blue) classifications are observed. This results in wandering trajectories during navigation in many scenarios. In some cases with grass and asphalt, both surfaces are considered to be equally traversable. TerraPN on the other hand, clearly prefers asphalt over grass. This reflects the discrepancy between obtaining labels from the robot-terrain interactions (in TerraPN), versus

human annotations (prior segmentation methods).

In the case with red tiles and concrete (row 3), TerraPN predicts both surfaces to have similar surface costs for the Husky robot, with high costs at the boundary between them (with soil). TerraPN accurately correlates the texture of the red tiles (previously unseen) with the tiles it has seen during training (similar to row 1 in Fig. 4.6). OCRNet predicts that the red tile surface is forbidden, and concrete to be moderately traversable. PSPNet is more accurate. However, it too has incorrect forbidden segmentation in the red tiles in some instances.

TerraPN sometimes does get confused. For example, in row 4, the different shades of the asphalt are considered to be different surfaces and slightly different costs are predicted (green versus blue). However, the small change in the predicted surface cost does not affect the robot’s trajectories.

Navigation Performance: We evaluate TerraPN’s navigation performance both quantitatively (Table 4.3) and qualitatively (Fig. 4.1 and 4.7) in our test scenarios. We do not report the metrics for scenario 5 since other methods mostly failed to reach the goal. Quantitatively, we observe that TerraPN leads to the highest success rates in all the scenarios. Other methods failed to reach the goal by either traversing surfaces where the robot got stuck such as rocks (scenario 2), or mud (scenario 3, 4). Or, in the cases of OCRnet and PSPnet, their trajectories wandered away from the goal due to the mis-classification of certain surfaces as forbidden.

In terms of trajectory lengths, TerraPN is comparable or lower to the navigation methods based on OCRnet [218] and PSPnet [219]. This is again due to wandering trajectories as explained before. DWA and TERP take shorter trajectories in some cases since they directly move toward the goal in the absence of obstacles and major elevation changes, without considering surface

variations. However, in the cases where the normalized length is less than 1, DWA and TERP wrongly assumed that the robot had reached its goal due to odometry errors (due to wheel slip) while navigating with high velocities on high-cost surfaces. Although TerraPN's trajectories deviate to avoid high-cost surfaces, their normalized lengths are close to one. This is due to the influence of the *head()* term in the objective function.

TerraPN also outperforms the other methods in terms of vibration costs in three scenarios since its trajectories were on smooth, low-cost surfaces as much as possible (see Fig. 4.1 and 4.7) with appropriate speeds. In scenario 2, TERP outperforms TerraPN due to the presence of concrete curbs. TerraPN could not distinguish the elevation changes between the ground and the curb, and moved over its edge near the rocks (see Fig. 4.7b) in some trials. Since TERP uses elevation maps to compute cost maps, it takes an overly conservative trajectory to avoid the curb. OCRNet and PSPNet have higher vibration costs although they possess semantic information about the terrains due to their longer, meandering trajectories.

In terms of mean velocity, we observe that TerraPN varies the robot's speed when navigating on different surfaces in the various scenarios (see yellow and red trajectories in Figs. 4.1, 4.7). All other methods move towards the robot's goal close to its maximum velocity (0.6 m/s). TerraPN's overall lower mean velocity varies depending on the number and type of surfaces in the scenario (which also reflects on the vibration cost). The regions where TerraPN slowed down the robots can be seen in Figs. 4.1 and 4.7. In the highly unstructured scenario 5, only TERP and TerraPN reached the robot's goal. TerraPN took a slow, cautious speed towards the robot's goal while TERP took a slow winding path based on the elevation changes it sensed in the terrain. DWA moved fast directly toward the goal and stopped short of it due to odometry errors caused due to the surface's poor traction.

Inference and Training Times: Table 4.2 compares TerraPN’s cost prediction inference time with various semantic segmentation methods trained for outdoor environments. TerraPN’s inference time when using a uniform patch sampling of 50×50 patches, is comparable to OCRNet and PSPNet. However, our non-uniform patch sampling when using patches of side lengths 50, 100, and 200 reduces the inference time by 47.27% compared to uniform sampling. For comparison, we also include CGNet [220], an extremely lightweight ($< 500K$ parameters) segmentation network.

However, TerraPN’s cost prediction network trains in $\sim 20 - 25$ minutes to achieve a performance suitable for navigation compared to $\sim 3 - 10$ hours need to train other networks for satisfactory segmentation accuracy. Additionally, using TerraPN’s cost map leads to superior navigational performance, as observed before.

4.3.5 Limitations

TerraPN must navigate on a surface to estimate the corresponding navigability costs. This strategy cannot be applied on completely non-traversable surfaces (e.g. swampy or densely vegetated terrains). Significant lighting changes in the environment could adversely affect the surface cost prediction. Further analysis on the number of surfaces our method can learn effectively must be conducted. TerraPN cannot distinguish subtle elevation changes in the same or similar surfaces due to the unavailability of depth and elevation inputs to the system.

Since TerraPN is fundamentally formulated for a wheeled robot, it is limited by the robot’s dynamics. Therefore, it cannot be directly applied to deal with dense outdoor vegetation since wheeled robots have a high probability of getting entrapped in such settings. VERN, presented

in the next section, is formulated to navigate a legged robot through dense vegetation.

4.4 VERN

In this section, we provide some background (section 4.4.1) on the concepts used in VERN. Next, we explain VERN’s perception formulation in section 4.4.2 and its navigation formulation in section 4.4.3, elaborate the details of its implementation and evaluations in section 4.4.4, and discuss its limitations in section 4.4.5.

4.4.1 Preliminaries

In this section, we provide our problem formulation, and briefly explain siamese networks used for vegetation classification. For navigation, VERN extended DWA’s formulation explained in section 4.3.1.2. Our overall system architecture is shown in Fig. 5.3.

4.4.1.1 Setup and Conventions

We assume a legged robot setup with holonomic dynamics equipped with a 3D LiDAR (that provides point clouds and 2D laser scans) and an RGB camera. Certain quantities are measured relative to a global coordinate frame (referred to as odom hereafter) with the robot’s starting location as its origin. Some quantities are relative to a cost map grid moving with and centered at the robot. The quantities can be transformed between these frames using transformation matrices T_C^O or T_O^C . The reference frame for each quantity is indicated in the superscript (O or C) of its symbol. We denote vectors in bold, and use symbols such as $i, j, \text{row}, \text{col}$ as indices. Apart from its pose w.r.t odom, the robot does not have access to any global information.

4.4.1.2 Problem Domain

We consider navigation in environments containing three kinds of vegetation together: 1. Tall grass, 2. Bushes/shrubs, and 3. Trees. We make the following assumptions about the three types.

Properties of Vegetation: Tall grass is pliable (therefore traversable), with height $> 0.3m$ (could be taller than the height the robot's sensors are mounted at), and could be of varying density in the environment. Bushes/shrubs (height $0.1 - 0.5m$), and trees (height $> 2m$) are non-pliable/untraversable and must be circumvented.

Adverse Phenomena: Navigating through such vegetation, the robot could encounter three adverse phenomena: 1. Freezing, 2. Physical entrapment in vegetation, and 3. Collisions. The conditions for these phenomena are,

1. Freezing: $V_r = \emptyset$,
2. Entrapment: $(v^*, \omega^*) \neq (0, 0)$ but $\Delta x_{rob}^O = \Delta y_{rob}^O = \Delta \theta_{rob}^O = 0$,
3. Collisions: $dist(\text{robot}, \{\text{bushes/trees}\}) = 0$.

Here, V_r refers to the robot's feasible, collision-free velocity space. (v^*, ω^*) is the velocity commanded by the robot's planner, and $\Delta x_{rob}^O, \Delta y_{rob}^O, \Delta \theta_{rob}^O$ refer to the robot's change in position and yaw orientation relative to the global odom frame. Freezing occurs in densely vegetated environments if the planner cannot find any collision-free velocity to reach its goal. To avoid this, the robot must accurately identify pliable vegetation to navigate through. The robot could also get physically entrapped in dense vegetation if its legs are caught in grass, small branches of

bushes, etc. Finally, the robot could collide if its planner misclassifies a non-pliable obstacle as pliable.

With these definitions, VERN’s problem formulation can be stated as follows: To compute a robot velocity (v^*, ω^*) such that $traj^O(v^*, \omega^*) \in Free^O \cup Grass^O$, $traj^O(v^*, \omega^*) \notin Tree^O \cup Bush^O$, assuming that $Tree^O \cap Grass^O = Tree^O \cap Bush^O = Bush^O \cap Grass^O = \emptyset$.

Here, $Tree^O, Bush^O, Grass^O$ represent the sets of locations containing trees, bushes, and grass, respectively that were detected by the robot w.r.t the odom frame. $Free^O$ represents free space locations. $traj^O()$ returns the forward-simulated trajectory of a (v, ω) pair relative to the odom frame. We assume that the robot can balance and walk on all the terrains we consider. We also assume that the robot never encounters a scene with no free space and only non-pliable vegetation (an unresolvable scenario).

4.4.1.3 Siamese Networks

Siamese networks [221] are a class of neural networks used for one-shot or few-shot learning (learning with very few samples). They use two sub-networks with identical network structures, parameters, and weights, and are connected at the end using an energy function (e.g., L2 norm). They accept two distinct images as inputs for the two sub-networks and are trained to map similar images close to each other in a low-dimensional latent space. Dissimilar images have latent representations that are mapped far away from each other.

Siamese networks are typically used for verification, and identification tasks (e.g., facial recognition) where the number of training images is low. Moreover, since they are trained to identify the similarity between images, the training dataset does not require any human annotation.

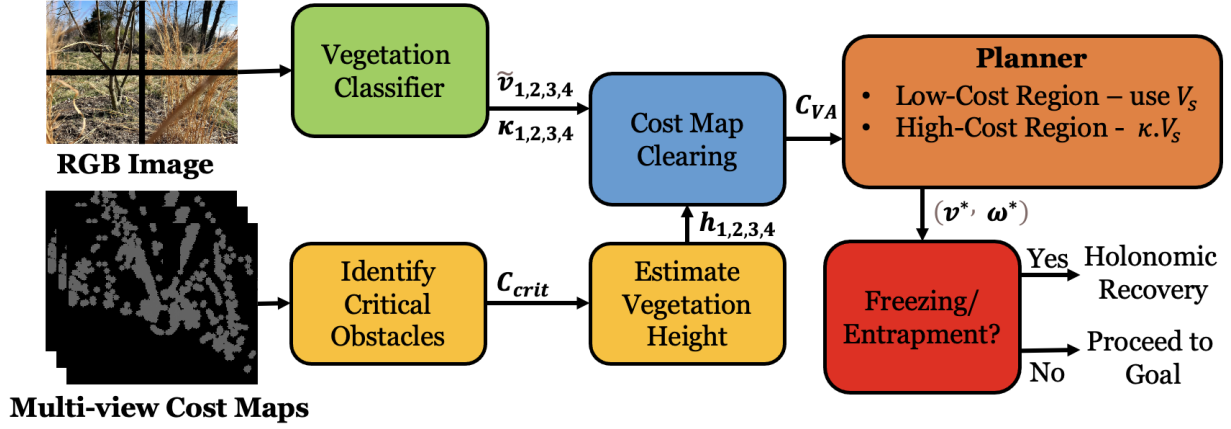


Figure 4.8: VERN’s overall system architecture for vegetation classification to estimate pliability/traversability. Our model uses classification, its confidence, and vegetation height to create a vegetation-aware cost map C_{VA} used for planning. In high traversability cost regions, our planner executes cautious behaviors by limiting the robot’s velocity space. If the robot freezes or gets entrapped, it executes holonomic recovery behaviors.

Using them allows our classifier to be trained with a few hundred of images corresponding to each class. Our classifier’s network architecture using siamese networks is shown in Fig. 4.9.

4.4.2 VERN: Vegetation Classification

In this section, we first explain VERN’s perception components that train a vegetation classifier and integrate it with a cost map.

Our vegetation classifier uses an RGB image (I_t^{RGB}) obtained from a camera on the robot at a time t as input. Although a plant’s structure is well preserved in an image, using the entire image for classification is infeasible because the scene in I_t^{RGB} typically contains two or more types of vegetation together. Therefore, we split I_t^{RGB} into four quadrants Q_1 (top-left), Q_2 (top-right), Q_3 (bottom-left), Q_4 (bottom-right) as shown in Figs. 4.9, 4.10. A quadrant predominantly contains a single type of vegetation typically (see Fig. 4.10).

We classify vegetation in the quadrants into four classes: 1. sparse grass, 2. dense grass, 3.

bush, and 4. tree. We separate sparse and dense grass due to their visual dissimilarity.

4.4.2.1 Data Preparation

To create the training dataset, images collected from different environments are split into quadrants and grouped together manually based on the vegetation type predominant in a quadrant. Next, pairs of images are created either from the same group or from different groups. The pair with similar images (same group) is automatically labeled 1, or 0 otherwise. The entire data preparation process takes about 30-45 minutes of manual effort. The obtained image pairs and the corresponding labels are passed into the classification network for training.

4.4.2.2 Network Architecture

Our vegetation classifier network consists of two identical feature extraction branches to identify the similarity between input image pairs. Our feature extraction branches are based on the MobileNetv3 [222] backbone. We choose MobileNetv3 because it incorporates depth-wise separable convolutions (i.e., fewer parameters), leading to a comparatively lightweight and fast neural network. The outputs of the MobileNetv3 branches (h_1 and h_2) are one-dimensional latent feature vectors of the corresponding input images. The Euclidean distance between the two feature vectors is calculated and passed through a *sigmoid* activation layer to obtain the predictions.

We utilize the *contrastive loss* function, which is capable of learning discriminative features to evaluate our model during training. Let $\hat{y}$ be the prediction output from the model. The contrastive loss function J is:

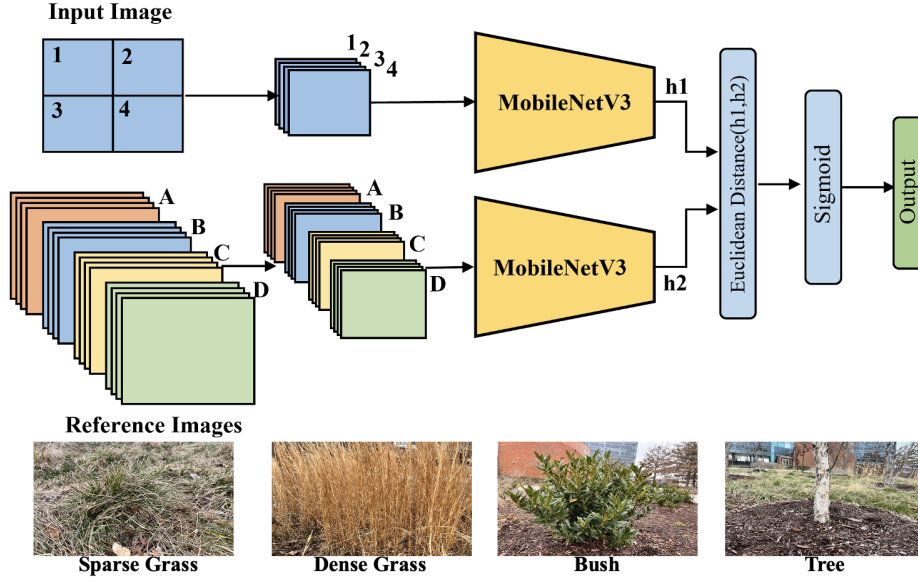


Figure 4.9: VERN's classification network architecture. Quadrants of the real-time camera image are paired with several reference images for each class and fed into the two identical branches of our network. Example reference images for each class are shown at the bottom.

$$J = \hat{y} \cdot d^2 + (1 - \hat{y}) \cdot \max(\text{margin} - d, 0)^2, \quad (4.15)$$

where d is the euclidean distance between the feature vectors h_1 and h_2 . *Margin* is used to ensure that dissimilar image pairs are at least *margin* distance apart.

4.4.2.3 Network Outputs

During run-time, the quadrants Q_1, Q_2, Q_3 , and Q_4 are each paired with several reference images of the four classes. The several reference images per class have different viewpoints and lighting conditions. They are fed into the classifier model $\mathcal{F}$ as a batch to obtain predictions as,

$$\mathcal{F}(Q_1, Q_2, Q_3, Q_4) = \tilde{V}_{4 \times 4} \mid (\tilde{V}_{i,j} \in [0, 1]) \ i, j \in \{1, 2, 3, 4\}. \quad (4.16)$$

$\tilde{V}_{4 \times 4}$ is the output prediction matrix whose values $\tilde{V}_{i,j}$ correspond to the *least* euclidean distance of the quadrant Q_i from any of the reference images for the j^{th} class. The closer $\tilde{V}_{i,j}$ is to 0, the

higher the similarity between Q_i and class j .

We extract two outputs from $\tilde{V}_{4 \times 4}$. Namely, for each quadrant Q_i : 1. the vegetation class that is most similar to it ($\tilde{v}_i$), and 2. the corresponding similarity score (d_i):

$$\tilde{v}_i = \underset{j}{\operatorname{argmin}} \tilde{V}_{i,j}, \text{ and } d_i = \tilde{V}_{i,j}. \quad (4.17)$$

For readability, hereafter we denote $\tilde{v}_i$ as belonging to the Pliable Vegetation (PV) set if $\tilde{v}_i = 1$ or 2 (sparse/dense grass), and to the Non-Pliable Vegetation (NPV) set if $\tilde{v}_i = 3$ or 4 (bush/tree).

We define the confidence of classification κ_i as,

$$\kappa_i = e^{-\alpha \cdot d_i}, \quad (4.18)$$

where α is a tunable parameter. We observe that $d_i \rightarrow 0 \implies \kappa_i \rightarrow 1$, and vice versa.

4.4.3 VERN: Navigation in Dense Vegetation

We use occupancy grids/cost maps generated using 2D LiDAR scans to detect obstacles around the robot. Our cost map can be defined as,

$$C_z(row, col) = \{p \mid p \in \{0, 100\}\}. \quad (4.19)$$

Here, z is the height (along the robot's Z-axis) at which the 2D scan is recorded on a plane parallel to the ground. p is a binary variable representing if an obstacle is present at (row, col) of the cost map. In a densely vegetated environment, C_z contains obstacles in all the locations where the LiDAR's scans have a finite proximity value. However, it does not account for the pliability of certain types of vegetation, which makes them passable and therefore not true obstacles.

Therefore, we use the classification results, its confidence (section 4.4.2.3), and estimated vegetation height to augment our cost map prior to navigation.

4.4.3.1 Multi-view Cost Maps

To estimate the height of the environmental obstacles, we use three cost maps of equal dimensions C_{low} , C_{mid} , and C_{high} corresponding to 2D scans from three different heights in 3D LiDAR's point clouds. The cost maps correspond to a height lower, equal to, and higher than the height at which the robot's 3D LiDAR is mounted, respectively. C_{low} contains obstacles of all heights around the robot. C_{mid} and C_{high} contain taller obstacles such as tall grass, trees, walls, humans, etc. Using only three views instead of projecting the entire point cloud onto a 2D plane reduces the computation burden. It also helps identify truly tall obstacles and avoids misclassifying overhanging leaves from trees as tall obstacles.

We consider tall obstacles as *critical obstacles* due to the high probability of them being solid obstacles such as trees and walls. To identify critical obstacles, we perform the element-wise sum operation as follows,

$$C_{crit} = C_{high} \oplus C_{mid} \oplus C_{low}. \quad (4.20)$$

$C_{crit}(row, col) \in \{0, 100, 200, 300\}$, and regions with higher costs contain critical obstacles.

4.4.3.2 Vegetation-aware Cost Map

To combine $\tilde{v}_i$, κ_i and C_{crit} , we must correlate the regions viewed by quadrants $Q_{1,2,3,4}$ in I_t^{RGB} with the corresponding regions in C_{crit} .

Homography: To this end, we apply a homography transformation H to project I_t^{RGB} onto the cost map. Let $reg()$ denote a function that returns the real-world (x, y) coordinates corresponding to a region of a cost map or image. We obtain the image quadrant relative to the cost map (see green/red rectangles in Fig. 4.10 bottom) as,

$$Q_i^C = \{(row, col) | reg(C(row, col)) = reg(H(Q_i))\}. \quad (4.21)$$

Cost Map Clearing: We now use $\tilde{v}_i$, κ_i , C_{crit} , and Q_i^C to clear/modify the costs of the grids in C_{low} . We choose to modify and plan over C_{low} since it detects obstacles of all heights. First, we calculate the normalized height measure (between 0 to $\pi/2$) of the obstacles in each quadrant i using C_{crit} as,

$$h_i = mean(C_{crit}(Q_i^C)) / c_{max} \cdot \frac{\pi}{2}, \quad (4.22)$$

where, $c_{max} = 300$ the maximum value in C_{crit} . Next, in each quadrant Q_i^C of C_{low} we modify the cost as,

$$C_{VA}(Q_i^C) = C_{low}(Q_i^C) \cdot \frac{clear(\kappa_i, h_i)}{max(clear(\kappa_i, h_i))} \quad (4.23)$$

$$clear() = \begin{cases} w_{s \text{ or } d} \cdot (1 - \kappa_i) + \frac{2 \cdot h_i}{\pi} & \text{if } \tilde{v}_i \text{ is PV} \\ (w_{NPV} \cdot \kappa_i + b_{NPV}) + \sin(h_i) & \text{if } \tilde{v}_i \text{ is NPV.} \end{cases} \quad (4.24)$$

Here, C_{VA} is a vegetation-aware cost map, w_s , w_d , w_{NPV} , and b_{NPV} are positive constants satisfying the condition $w_d > w_s$, and $b_{NPV} > w_d + 1$. The weights w_s and w_d are used when $\tilde{v}_i$ is sparse and dense grass respectively. We incorporate $\sin(.)$ for NPV to ensure that the significantly tall obstacles have higher costs and differentiable cost values from short obstacles. In contrast, we

consider a linearly varying cost function w.r.t. the vegetation height for PV since the resistance to the robot from pliable tall objects is correlated with their height. We note that $\sin(h_i) \geq \frac{2 \cdot h_i}{\pi}$ for $h_i \in [0, \pi/2]$.

For PV classifications, low confidence and tall vegetation lead to higher navigation costs. Intuitively, this leads to the robot preferring to navigate through high-confidence, and short pliable vegetation whenever possible. Conversely, for NPV classifications, high confidence, and tall vegetation lead to higher costs since such regions should definitely be avoided. Accounting for low-confidence classifications in the formulation helps handle misclassifications and assign costs accordingly.

Proposition 4.4.1. *The $clear()$ function modifies C_{low} such that traversability costs in PV regions are always lower than in NPV regions.*

Proof. The maximum cost for PV is $w_d + 1$ (when $\kappa_i \rightarrow 0$ and $h_i \rightarrow \pi/2$), and the minimum cost for NPV is b_{NPV} (when $\kappa_i \rightarrow 0$ and $h_i \rightarrow 0$). Conditions $w_s < w_d$, and $b_{NPV} > w_d + 1 \implies \max \text{cost of PV} < \min \text{cost of NPV}$. Therefore, in the absence of free space, the robot always navigates through PV and avoids NPV regions. Therefore, regions with PV will always be preferred for navigation. ■

4.4.3.3 Cautious Navigation

We adapt DWA (section 4.3.1.2) to use our vegetation-aware cost map for robot navigation. We calculate the obstacle cost ($obs(.)$) associated with every (v, ω) pair by projecting its predicted trajectory $traj^C(v, \omega)$ relative to the cost map over C_{VA} and summing as,

$$obs(v, \omega) = \sum_{(row, col) \in traj^C(v, \omega)} C_{VA}(row, col). \quad (4.25)$$

Next, we compute the total cost $Q(v, \omega)$ (equation 4.1). The (v, ω) that minimizes this cost is used for navigation. In some cases, the robot might have to navigate a region with high traversability cost (represented say in Q_i^C) which typically occurs with NPV. To imbibe cautious navigation behaviors for such scenarios, we stunt the robot's complete velocity space as $V_s = [[0, \kappa_i \cdot v_{max}], \kappa_i \cdot [-\omega_{max}, \omega_{max}]]$.

4.4.3.4 Recovery Behaviors

In highly dense vegetation or in regions with low confidence classifications, the robot could still freeze or get physically entrapped. We observe that Spot's high degrees of freedom and superior dynamics allow it to recover itself from such situations if it avoids rotational motions. Therefore, we propose holonomic recovery behaviors to recover Spot from such situations. To this end, the robot periodically stores *safe*^O locations relative to the odom frame whenever the conditions for freezing or entrapment (section 4.4.1.2) are not satisfied. If the conditions are satisfied, the robot stores its current location into an *unsafe*^O location list and chooses a safe location such that,

$$unsafe^C = T_O^C \cdot unsafe^O, \quad (4.26)$$

$$C_{VA}(unsafe^C) = \infty, \quad (4.27)$$

$$\mathbf{p}_{rec}^O = \underset{\mathbf{p}^O \in safe}{\operatorname{argmin}} \left(dist(\mathbf{g}^O, \mathbf{p}^O) \right), s.t \ (\mathbf{P}^O - \mathbf{P}_{rob}^O) \in Free^O \cap Grass^O, \quad (4.28)$$

$$[v_x, v_y] = k_p \cdot [\mathbf{p}_{rec} - \mathbf{p}_{rob}]. \quad (4.29)$$

Here, $\mathbf{p}_{rec}^O$, $\mathbf{g}^O$, and $\mathbf{p}_{rob}^O$ are the safe location chosen to recover to, the robot's goal and current

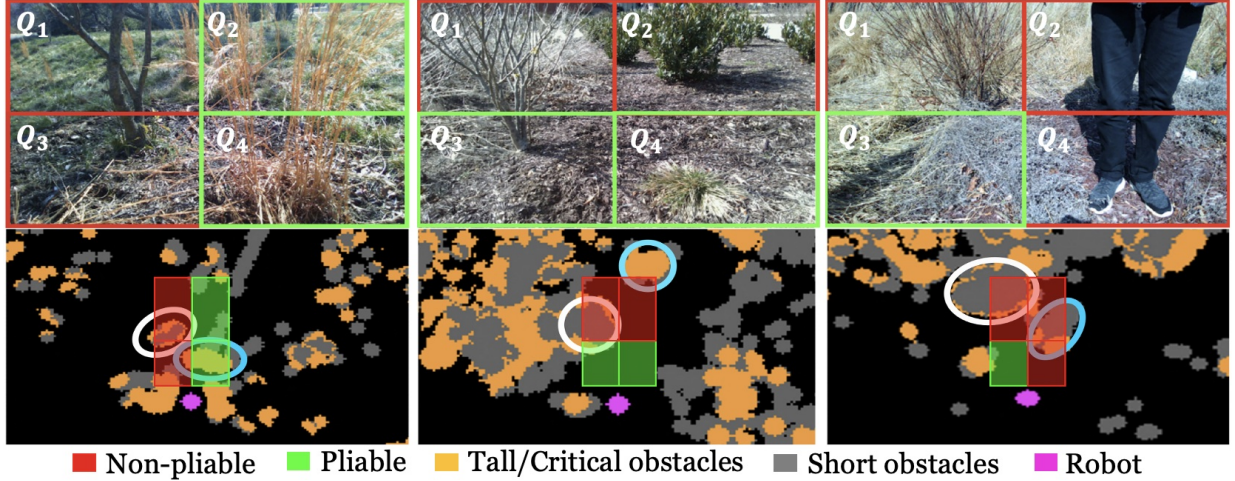


Figure 4.10: Snapshots of the vegetation classification results on RGB images (top), and the corresponding cost maps C_{VA} marked with critical obstacles (bottom) from trials in scenarios 2 (left), 3 (center), and 4 (right). A green/red rectangle in a quadrant $Q_{1,2,3,4}$ in the RGB image represents a PV/NPV classification respectively. The same quadrants are projected onto the cost maps ($Q_{1,2,3,4}^C$) along with their classifications for visualization. The cost for the vegetation within the green regions is reduced significantly using equation 4.24. The encircled regions in each cost map correspond to an obstacle in the RGB image. **[Left]**: White - Tree, Blue - Tall grass, **[Center]**: White - Tree, Blue - Bush, **[Right]**: White - Tree, Blue - Human. Our classifier accurately detects trees, and bushes as non-pliable/untraversable, and tall grass as traversable. In scenario 4, it accurately detects the human as untraversable due to its dissimilarity with all training classes and tall height.

location. The condition in equation 4.28 ensures that the line connecting the robot and the recovery point only lies in traversable regions. Once the robot recovers, it proceeds to its goal after marking the *unsafe* location with high costs in C_{VA} to circumvent it.

4.4.4 Results and Analysis

We detail our method’s implementation and experiments conducted on a real robot. Then, we perform ablation studies and comparisons to highlight VERN’s benefits.

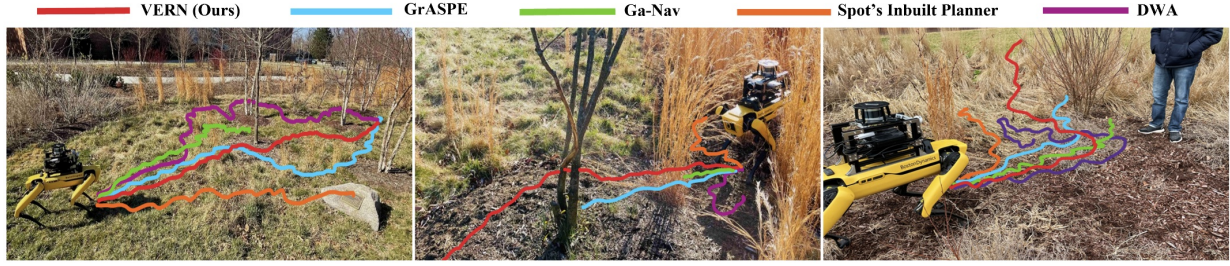


Figure 4.11: Spot robot navigating in: **[Left]** Scenario 1, **[Center]** Scenario 2, and **[Right]** Scenario 4. We observe that VERN navigates through pliable/traversable vegetation in the absence of free space. This leads to significantly higher success rates, low freezing rates, and trajectory lengths. Other methods either freeze or take long, meandering trajectories to the goal.

4.4.4.1 Implementation and Dataset

VERN’s classifier is implemented using Tensorflow 2.0. It is trained in a workstation with an Intel Xeon 3.6 GHz processor and an Nvidia Titan GPU using real-world data collected from a manually teleoperated Spot robot. The robot is equipped with an Intel NUC 11 (a mini-PC with Intel i7 CPU and NVIDIA RTX 2060 GPU), a Velodyne VLP16 LiDAR, and an Intel RealSense L515 camera. The training images (~ 600 per class) were collected from various environments in the University of Maryland, College Park campus. Our model takes about 6 hours to train for 55 epochs.

4.4.4.2 Evaluations

We use the following evaluation metrics to compare VERN’s performance against several navigation methods: 1. Boston Dynamics’ in-built autonomy on Spot, 2. DWA [177], 3. GA-Nav [172], and 4. GrASPE [176]. Spot’s in-built autonomy uses its stereo cameras in four directions around the robot, estimates obstacles, and navigates to a goal. DWA is a local planner that utilizes a 2D LiDAR scan for obstacle avoidance. GA-Nav combines semantic segmentation

for terrain traversability estimation with elevation maps for outdoor navigation. It is trained on publically available image datasets (RUGD [223] and RELIS-3D [224]). GrASPE is a multi-modal fusion framework that estimates perception sensor reliability to plan paths through unstructured vegetation. We further compare VERN without height estimation and recovery behaviors. The metrics we use for evaluations are,

- **Success Rate** - The number of successful goal-reaching attempts (while avoiding non-pliable vegetation and collisions) over the total number of trials.
- **Freezing Rate** - The number of times the robot got stuck or started oscillating for more than 5 seconds while avoiding obstacles over the total number of attempts. Lower values are better.
- **Normalized Trajectory Length** - The ratio between the robot’s trajectory length and the straight-line distance to the goal in all the successful trajectories.
- **False Positive Rate (FPR)** - The ratio between the number of false positive predictions (i.e., actually untraversable/non-pliable obstacles predicted as traversable) and the total number of actual negative (untraversable) obstacles encountered during a trial. We report the average over all the trials.

If the sum of the success and freezing rates do not equal 100, it indicates that the robot has collided in those cases. We also compare these methods qualitatively using the trajectories pursued while navigating. For perception comparisons, we quantitatively evaluate the accuracy and F-score of MobileNetv3 [222], EfficientNet [225], and Vision Transformer [226] when trained and evaluated on our dataset. We also show VERN’s classification results and cost map clearing in Fig. 4.10.

4.4.4.3 Navigation Test Scenarios

We compare the performance of all the navigation methods in the following real-world outdoor scenarios (see Figs. 4.2 and 4.11) that differ from the training environments. The scenarios are described in increasing order of difficulty. Ten trials are conducted in each scenario for each method.

- **Scenario 1** - Contains sparse tall grass, and trees.
- **Scenario 2** - Contains trees and dense tall grass in close proximity. The robot must identify grass and pass through it to reach its goal successfully.
- **Scenario 3** - Contains trees, bushes, and dense grass in close proximity. The robot must identify grass and pass through it. See Fig. 4.2
- **Scenario 4** - Contains dense grass, trees, and an obstacle (human) unseen during training.

4.4.4.4 Analysis and Discussion

Classification Accuracy: We observe from Table 4.4 that MobileNetv3 has the best accuracy and F-score compared to other methods. EfficientNet is difficult to train and tune especially due to longer training time requirements. Similarly, ViT requires a lot more data for training and is not compatible with the Siamese network framework.

Navigation Comparison: The quantitative navigation results are presented in Table 4.3. We observe that VERN's (with height estimation used in equation 4.24 and recovery behaviors) success rates are significantly higher compared to the other methods. Spot's in-built planner and

Metrics	Method	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Success Rate (%) (Higher is better)	Spot's Inbuilt Planner	50	0	0	0
	DWA [177]	80	0	0	0
	GA-Nav [172]	60	20	30	30
	GrASPE [176]	80	60	50	40
	VERN w/o height estimation	80	60	40	30
	VERN w/o recovery behavior	100	60	40	40
	VERN(ours)	100	90	70	70
Freezing Rate (%) (Lower is better)	Spot's Inbuilt Planner	30	100	100	90
	DWA [177]	20	100	100	100
	GA-Nav [172]	40	80	70	70
	GrASPE [176]	10	20	20	50
	VERN w/o height estimation	20	40	40	70
	VERN w/o recovery behavior	0	40	60	60
	VERN(ours)	0	10	20	30
Norm. Traj. Length (Closer to 1 is better)	Spot's Inbuilt Planner	1.23	0.37	0.24	0.22
	DWA [177]	1.52	0.46	0.34	0.62
	GA-Nav [172]	1.31	1.39	1.32	1.49
	GrASPE [176]	1.18	1.09	1.46	1.37
	VERN w/o height estimation	1.39	1.41	1.35	1.30
	VERN w/o recovery behavior	1.42	1.48	1.46	1.39
	VERN(ours)	1.11	1.19	1.28	1.23
False Positive Rate	GA-Nav [172]	0.33	0.39	0.30	0.61
	GrASPE [176]	0.25	0.31	0.28	0.44
	EfficientNet [225]	0.28	0.23	0.32	0.35
	VERN(ours)	0.15	0.18	0.12	0.21

Table 4.3: Navigation performance of our method compared to other methods on various metrics. VERN outperforms other methods consistently in terms of success rate, freezing rate, false positive rate, and normalized trajectory length in different unstructured outdoor vegetation scenarios.

DWA consider all vegetation as obstacles. This leads to Spot becoming unstable or crashing when using its in-built planner and leads to excessive freezing and oscillations when using DWA.

GA-Nav considers all vegetation (except trees) as partially traversable because it segments most flora as grass. It also cannot differentiate vegetation due to the lack of such precisely human-annotated datasets. Additionally, in cases where vegetation like tall grass partially occludes the RGB image (see scenario 2 in Fig. 4.11 [center]), GA-Nav struggles to produce segmentation results leading to the robot freezing or oscillating. In regions with tall grass, GA-Nav's elevation maps also become erroneous.

Methods	Accuracy	F-score
MobileNetv3 [222]	0.957	0.833
EfficientNet [225]	0.758	0.632
Vision Transformer (ViT) [226]	0.689	0.546

Table 4.4: The accuracy and F-scores (higher values are better) of three feature-extracting backbones used to train our dataset. We observe that our MobileNetv3 has the best accuracy and F-score because of its depth-wise separable convolutions. This leads to faster and better learning.

GrASPE performs the best out of all the existing methods. It is capable of passing through sparse grass and avoiding untraversable obstacles. However, in highly dense grass that is in close proximity to trees, GrASPE is unable to accurately detect and react quickly to avoid collisions with trees.

Since some existing methods do not reach the goal even once, their trajectory lengths are less than 1. The values are still reported to give a measure of their progress toward the goal. Spot’s in-built planner progress the least before the robot froze or crashed. In some cases, the methods take meandering trajectories (e.g. DWA) in scenario 1 before reaching the goal. Notably, VERN deviates the least from the robot’s goal and that reflects in the low trajectory length.

FPR: We compare VERN’s vegetation classifier’s false positive rate (FPR) with GA-Nav and the EfficientNet-based classifier using manual ground truth labeling of the vegetation in the trials. We observe that GA-Nav leads to a significantly high FPR in all four scenarios primarily because GA-Nav’s terrain segmentation predictions are trained on the RUGD dataset. Moreover, GA-Nav’s incorrect segmentation under varying lighting conditions and occlusions increases the false positive predictions. GrASPE and the EfficientNet-based classifier are trained using the same images we used to train VERN. However, we observe that their accuracy is comparatively lower than our VERN model. This is primarily due to the fine-grained feature learning capabilities of VERN’s MobileNetv3-based backbone. Additionally, GrASPE’s predictions lead to erroneous

results when its 3D point cloud cannot identify the geometry of the vegetation such as trees and bushes under visually cluttered instances.

Ablation study: We compared VERN and its variants without using height estimation, and recovery behaviors. We observe that when the height estimation is not used for cost map clearing, the robot’s success rate drops. This is mainly because height helps differentiate short and tall pliable vegetation (where the robot could freeze). Additionally, estimating the critical regions based on the height helps avoid obstacles such as humans (Scenario 4) which are not part of our classifier.

Using recovery behaviors helps reduce freezing in the presence of dense obstacles (especially in scenarios 2, 3, and 4). Additionally, moving the robot to a safe location, and marking and remembering the unsafe region allows the robot to preemptively avoid the region in subsequent trials.

4.4.5 Limitations

Our primary navigation assumes non-holonomic robot dynamics to utilize DWA’s dynamic feasibility guarantees. However, the legged robot dynamics are holonomic, and further investigation is required to extend our method to relax its action constraints. We assume that the different kinds of vegetation are not intertwined with one another. This may not be the case in highly forested environments. VERN could lead to collisions in the presence of thin obstacles such as branches that are not detected in the cost maps.

4.5 Summary

In this chapter, we presented two methods for navigating unstructured terrains by accounting for their surface properties, and dense vegetation, by estimating its pliability. Both methods use perception models based on DNNs, and require no or minimal human labeling.

In the first part of this chapter, we presented TerraPN, an approach that used self-supervised learning to identify the surface characteristics in complex outdoor environments to perform autonomous robot navigation. Our method incorporated RGB images of surfaces and the robot’s velocities as inputs, and the IMU vibrations and odometry errors encountered by the robot while traversing on a surface as labels for self-supervision. The trained self-supervised network generated a surface navigability cost map that differentiated smooth, high-traction surfaces from bumpy, deformable surfaces. We introduced a novel navigation algorithm that accounts for the surface cost, computes cost-based acceleration limits for the robot, and computes dynamically feasible and collision-free trajectories.

In the second part of this chapter, we presented VERN, an algorithm to navigate a legged robot in densely vegetated environments with various traversability. We utilized a few-shot learning classifier trained on a few hundred quadrants of RGB images to distinguish vegetation with different pliability with minimal human annotation. This classification model was combined with the estimated vegetation height (from multiple local cost maps), and classification confidence using a novel cost map clearing scheme. Using the resulting vegetation-aware cost map, we deployed a local planner with recovery behaviors to recover the robot if it freezes or gets entrapped in dense vegetation.

TerraPN and VERN were implemented on a real Clearpath Husky and a Boston Dynamics

Spot robot, respectively. TerraPN was evaluated and demonstrated in unstructured terrains with rocks, mulch, deformable muddy surfaces, etc. VERN was evaluated in several densely vegetated environments with tall grass, bushes, and trees in close proximity.

An important limitation of these methods is their dependence on well-lit conditions, since their perception is primarily RGB image based. TerraPN's self-supervised learning helps more accurately characterize terrain traversability. However, this requires the robot to first traverse these terrains during training, which is not possible for completely non-traversable terrains. VERN, assumes that different kinds of vegetation are separated enough to be visually detectable. Such limitations may be overcome with fusing information from multiple sensor modalities.

Chapter 5: Perceiving Complex Obstacles and Contexts in Indoor and Outdoor Environments

5.1 Overview

The methods discussed in chapters 2, 3, and 4 have been specifically formulated for a particular domain (indoor or outdoor), and challenge (crowds, unstructured terrains, vegetation). This chapter discusses methods that address the challenges in sensing, and navigation in both indoor and outdoor environments. We first discuss these challenges here.

While navigating indoor and outdoor settings, robots face a variety of complex obstacles of various sizes, densities/solidities, and opacities that are challenging in terms of the robot's perception, and navigation. For instance, contemporary indoor environments contain objects such as string/beaded curtains, transparent surfaces such as glass walls [17, 227], etc. Outdoor scenarios, on the other hand, have complex vegetation such as pliable tall grass, bushes, trees, etc. in close proximity to and intertwined with each other. A major challenge, and an important requirement for autonomous navigation, is differentiating *truly solid* and impassable obstacles (furniture, glass surfaces, bushes, trees, etc.) from obstacles that can be passed through (beaded curtains, tall grass, etc).

To first detect obstacles, mobile robots have predominantly used RGB and depth images

[228], 2D LiDAR scans [177], 3D point clouds [229], etc. The raw data from these sensors has been used to: (1). Estimating the proximity of objects in the robot’s vicinity; (2). Computing a variation of an occupancy grid [230] or a cost map representation that indicates both an obstacle’s size and distance from the robot; or (3). Segmenting the scene to assess obstacles’ size, distance, and semantic meaning [231, 232]. Although such approaches have been used to aid navigation, they may not work well in environments composed of thin, pliable/bendable, and transparent obstacles. For instance, time-of-flight sensors such as depth cameras and LiDARs tend to detect thin, pliable, and passable obstacles (e.g. string curtains, tall grass) as solid obstacles that the robot must avoid [233]. On the other hand, transparent objects such as glass remain undetected since the laser rays mostly pass through them, leading to collisions during navigation. Similarly, perception methods using RGB images may not work well in terms of detecting transparent objects and visually similar but structurally dissimilar vegetation. Moreover, they are severely affected by the environment’s lighting changes.

Another important aspect of autonomous navigation is generating behaviors that are apt for the robot’s current context/scenario. The socially-compliant indoor navigation methods of chapters 2 and 3, and the outdoor navigation methods in chapter 4 generate such apt behaviors given a challenging context. However, all these methods were formulated to exhibit specific behaviors in a single context, and do not exhibit new behaviors in novel contexts. For example, TerraPN cannot exhibit socially compliant behaviors, and Frozone cannot prefer to navigate on pavements instead of sand. The robot perceiving and understanding its current context, and adapting its behaviors accordingly (akin to humans) is the next step of *generalized* autonomous navigation.

Humans are able to effectively navigate all these environments because we identify the

surrounding scenario or *context* (e.g. indoor hallway, outdoor terrain, etc), and follow certain explicit (e.g. crossing roads at crosswalks) and implicit (e.g. keep to one side of a corridor [234, 235]) social rules, and preferences (e.g. walk on well-paved surfaces). Existing navigation and perception methods are limited in their applicability. Conversely, Large Language Models (LLMs) [236–240] that use only text inputs and subsequently large Vision Language Models (VLMs) that use both RGB images and text prompts as inputs [237, 238, 240, 241] have overcome this generalization limitation to a large extent. They have demonstrated impressive zero-shot classification [242, 243], semantic visual understanding [244], and logical reasoning capabilities [245] in various tasks involving embodied vision language navigation [246, 247], open vocabulary manipulation [248, 249], etc. These capabilities would be beneficial for perceiving and reasoning about complex indoor and outdoor environments during autonomous navigation.

However, there are a few key challenges in using large VLMs for navigating mobile robots. Due to their large model size and high memory and computational demands, these models cannot be run on robot-mounted edge processors. They typically require several seconds to process queries and the response time varies based on the size of the queries [250]. Therefore, current VLMs cannot be queried for safety-critical tasks such as real-time collision avoidance with dynamic obstacles. Additionally, queries should be succinct and account for the environment’s context to elicit quick and accurate responses.

To address the problems of detecting complex indoor and outdoor obstacles, and generating *context-aware* navigation behaviors, we propose the following methods:

- Multi-layer Intensity Maps (MIMs) (section 5.3, Fig. 5.1), a novel obstacle representation constructed from the intensities of the reflected points in a point cloud. In addition to an

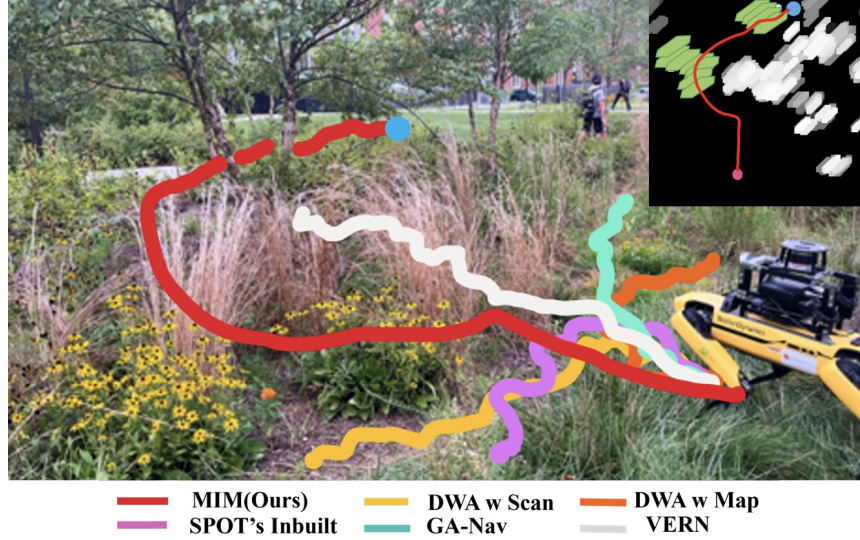


Figure 5.1: Comparison of trajectories while navigating using our Multi-layer Intensity Maps (MIM), DWA with laser scan [177], DWA with occupancy map [230], Spot’s Inbuilt Autonomy, and VERN [251] in complex vegetation. The MIM for the scenario is shown in the top right, with the robot in pink, its goal in blue, passable obstacles such as tall grass in green, and the robot’s trajectory overlayed for reference. MIMs help differentiate solid objects such as trees even when they are intertwined with tall grass. Other methods for robot perception in this case either freeze or collide with the solid obstacles.

object’s height and size, the multi-layer intensity map also reflects its true density/solidity.

MIMs can also be used to detect transparent obstacles such as glass, and thin passable objects such as string curtains indoors, and tall grass outdoors. We also present an adaptive inflation strategy that assesses the detected solid obstacles (e.g. concrete and glass walls, furniture indoors, and bushes and trees outdoors) to enlarge them in the multi-layer intensity map for efficient planning. Our strategy handles narrow scenarios such as doors, corridors, and passages between dense vegetation and trees, where existing navigation schemes freeze.

- CoNVOI (section 5.4, Fig. 5.2), a context-based navigation method that harnesses the zero-shot image classification, and logical reasoning capabilities of VLMs. CoNVOI uses a novel context-based prompting method that involves querying a compact VLM [242] to determine the robot’s environment context (e.g., indoor corridors, outdoor terrains, crosswalks).

CoNVOI then utilizes a predefined text prompt that describes a context-based navigation behavior and a marker-overlaid RGB image to a large VLM [237, 239, 240] to generate a reference trajectory for the robot. The marker overlaid image is created using a multi-modal visual marking method that enhances the RGB image inputs to a large VLM by overlaying visual markers/numbering on obstacle-free regions, aiding the VLM to focus on navigable regions, comprehending spatial relationships and underlying terrains across different regions within the image. Using CoNVOI’s context-based text prompts and marked images help generate reference trajectories that adhere to explicit and implicit social rules while also mimicking human preferences (e.g. walking on pavement, crossing at crosswalks, etc).

The rest of this chapter is organized as follows. In section 5.2, we discuss the related works in sensing complex indoor and outdoor environments, and the recent advances in VLMs that can bring context understanding and logical reasoning to robot navigation. Sections 5.3 and 5.4 explain MIMs’ methodology to sense challenging indoor and outdoor obstacles, and CoNVOI to generate context-aware navigation behaviors. We finally summarize this chapter in section 5.5.

5.2 Related Work

In this section, we survey the previous sensing methods for detecting complex indoor and outdoor obstacles and their limitations that MIM addresses, and the recent work in VLMs that forms the foundations for CoNVOI.



Figure 5.2: **[Top]**: The trajectories of a Spot robot crossing the road when using CoNVOI with GPT-4v [237] (in green), CoNVOI with Gemini [240] (in purple), teleoperated by a human (in red), GA-Nav [18], and DWA [29]. CoNVOI navigates the robot on the crosswalk by understanding the environmental context. **[Bottom]**: The trajectories when the Spot robot navigates to a goal beyond a blockade. While using CoNVOI, handling such scenarios can be added on the fly using a simple text prompt without any reformulation. The RGB images from the robot are shown with CoNVOI’s multi-modal visual marking (numbers in yellow). GPT-4v/Gemini is queried with the marked image and a context-based text prompt in quotes, and it returns the green reference path that follows explicit, and implicit social rules, and human-like preferences during indoor and outdoor navigation.

5.2.1 Sensing using Point Cloud Intensity

The importance of intensity information in LiDAR point cloud data has been an emerging focus in robotics [252–254]. Some methods investigate the use of intensity information alongside geometric features to enhance point cloud classification methods in outdoor settings [255]. These

methods highlight the potential of using the intensity information to provide a better understanding of obstacles, particularly when scene illumination is not consistent. Other methods include the ISHOT descriptor [256], which combines geometric and intensity data for improved place recognition. Lidar intensity maps have been also used for localization [257–259]. In [257], the authors present a robust Graph-SLAM framework that improves map accuracy for autonomous vehicles by encoding road surfaces based on LIDAR reflectivity. Moreover, the application of LiDAR intensity in visual navigation tasks has been explored. For instance, [260] introduces a LiDAR-intensity-image pipeline and demonstrates its performance in visual odometry (VO) and visual teach and repeat (VT&R) tasks. Lidar intensity maps have also been leveraged for various other applications, including orthoimage generation [261] and anisotropic surface detection [262].

5.2.2 Detecting Indoor Obstacles

Object detection in indoor settings has been widely studied for numerous applications including robot navigation, mapping, and computer graphics. Popular solutions in the literature include vision-based object detection and semantic segmentation approaches due to the structuredness of indoor environments. Moreover, the generation of necessary image datasets is feasible due to the limited diversity of indoor objects. However, detecting non-opaque objects such as glass remains a formidable challenge for vision-based systems due to the lack of visual clues. The method in [17] proposes GDNet, a glass detection network that identifies abundant contextual cues for glass detection using a large-field contextual feature integration (LCFI) module. UCTNet [231] proposes a cross-modal transformer network for indoor RGB-D semantic segmentation

to identify different objects such as curtains, doors, etc. However, such methods require large datasets with pixel-level ground truth labeling.

5.2.3 Detecting Outdoor Obstacles

Over recent years, there has been significant progress in the development of robotic systems designed for outdoor navigation [232,263–266]. One early approach can be found at [207], where the usage of laser measurements enabled navigation capabilities for robots such as detecting short and grass-like vegetation. However, this method is not universally applicable, particularly in complex, unstructured vegetative terrains. Complementary approaches have tackled associated issues in off-road navigation, specifically concerning varying slopes [27] and different terrains [267].

Many studies integrate proprioceptive with exteroceptive sensory data to enhance outdoor navigation [213,268,269]. Machine learning techniques have also been incorporated to augment the robot’s capabilities for navigating through pliable vegetative obstacles [156,175]. To this end, [251] uses a few shot-learning approaches to classify RGB vegetation images based on their traversability. This classifier is then integrated with a 3D LiDAR to construct a vegetation-aware traversability cost map.

5.2.4 Vision Language Models in Navigation

Over the past few years, LLMs [236] and VLMs [237–242] have demonstrated impressive accuracy in tasks such as scene/semantic understanding [270], grounding objects in a scene based on language description, instruction following, code generation [271], etc that are useful for robot

manipulation and navigation tasks. For instance, Shah et al. [247] utilized GPT-3 and CLIP to extract landmark descriptions from a text navigation instruction and ground them in images, respectively, to guide a robot’s navigation. Further, Shah et al. [272] demonstrated how the semantic understanding that large VLMs possess can help bias a robot’s exploration to search for a user-defined object/location. Similar to [247], Chen et al. [273] utilize LLMs to decompose navigation instructions into a series of actionable tasks that are executed by an action-aware navigation policy.

Zhu et al. [274] propose a system to use VLMs to describe a scene, and an LLM to verify if the scene matches a user-defined target. Visual Language Maps [275] demonstrate fusing VLM descriptions of landmarks with a 3D map that enables the robot to understand spatial references relative to landmarks. OK-Robot [248] vision-language representations computed from an environmental scan by VLMs are stored as semantic memory [243], and are matched with a language query to move to a desired target object and pick it up in a zero-shot manner.

Existing research has primarily used VLMs to identify navigation targets (*where* to navigate) [246, 247] and parse natural language commands but has not focused on the *how* (lower-level behaviors such as walking on pavements, not disturbing interacting pedestrians) to navigate robots to their targets. Additionally, while many studies have been evaluated in realistic simulated environments, those tested in real-world settings [247–249] have not addressed the use of semantic and contextual understanding of the environment to guide the robot’s navigation behaviors. CoNVOI’s goal is to leverage VLMs to guide low-level navigation behaviors to be the most suitable for a given scenario.

5.3 Multi-layer Intensity Maps

In this section, we explain the background definitions and assumptions (section 5.3.1) used in formulating MIMs, its methodology (section 5.3.2), evaluations (section 5.3.3), and limitations (section 5.3.4).

5.3.1 Background

In this section, we introduce and define the symbols and key concepts used in our work. Apart from these concepts, MIM’s motion planner is based on DWA (section 4.3.1.2)

5.3.1.1 Definitions and Assumptions

MIM’s formulation assumes that a sensor capable of generating 3D point clouds (e.g., 3D LiDAR, depth camera) is mounted on a robot with a 2D linear (v) and angular (ω) velocity space. Rigid coordinate frames are attached to the robot and sensor with the positive x, y, z directions facing forward, left, and upwards respectively, and for simplicity, we assume both frames to coincide. All positions and velocities are measured relative to these frames. At any time t , the robot has a preferred velocity direction aimed at its goal (g_x, g_y) as $\theta = \tan^{-1}(g_y/g_x)$.

Our approach is based on using the reflected intensities of point clouds that could be obtained from sensors such as 3D LiDARs, depth cameras, etc that have a laser source/transmitter and a receiver. We represent a point $\mathbf{p}$ in a point cloud as $\mathbf{p} = \{x, y, z, int\}$, where x, y, z denote the point’s location relative to the sensor, and $int \in [0, R]$ denotes its intensity. We define a 2D robot-centric grid map as containing $n \times n$ grids. Each grid is denoted by a row r and column c . Each grid represents a $g \times g$ area in the real world, and the value contained in it indicates the

probability of the presence of an obstacle. Finally, we use j and k to denote indices.

5.3.1.2 Point Cloud Intensity

Typically, a point's intensity is high ($int > 0.75R$) when it is reflected from solid, opaque, 3D (length, width, and height dimensions are not infinitesimal) objects since they prevent the sensor's laser rays from passing through, or scattering away from the sensor. In contrast, objects that are low density (e.g. tall grass which is a collection of thin blades of grass that scatter laser rays), and transparent (e.g. glass where laser rays mostly pass through) lead to low intensities ($int < 0.5R$) or in some cases, no intensity ($int = 0$).

5.3.1.3 Obstacle Properties

Our formulation leverages the property to accurately detect truly solid objects from the following categories defined based on how objects are sensed by existing perception modalities (2D LiDAR scan, RGB and depth images, etc) as:

- **True Positives (TP):** Solid, non-traversable objects detected as solid, e.g. walls, wooden furniture, etc.
- **True Negatives (TN):** Non-solid, traversable objects detected as passable or no obstacle, e.g. free space.
- **False Positives (FP):** Non-solid objects detected as solid, e.g. string/beaded curtains, pliable tall grass.
- **False Negatives (FN):** Solid objects detected as non-solid or as free space, e.g. transparent

objects.

5.3.1.4 Obstacle Inflation

Once a truly solid object is detected, it must be enlarged or *inflated* for the robot's planner to ensure that it avoids it at a safe distance [276]. Inflation is performed prior to planning to expand obstacles uniformly in all dimensions by a certain amount (typically the robot's radius, or maximum(length, width) to ensure that the planner avoids obstacles by a safe distance. Standard methods for obstacle inflation include performing the Minkowski sum [277] between the robot's radius and the obstacle, cost propagation from the obstacle, dilating obstacles using convolutions, etc. on a grid map. With these preliminaries, we state our problem formulation as follows:

To construct an $n \times n \times m$ grid map representation I_{ML}^t of obstacles from points $\mathbf{p} = \{x, y, z, int\}$ and classify each grid $(r, c) \in I_{ML}^t$ as a true positive (TP), false positive (FP), false negative (FN) obstacle, or true negative (TN) free space and enlarge TP and FN obstacles adaptively based on the robot's preferred velocity direction.

5.3.2 MIM: Approach

In this section, we discuss how 3D point cloud intensities can be used to construct the multiple 2D grid map layers of an MIM. The input point clouds could be obtained from any sensor such as a 3D LiDAR or a depth camera that also measures the intensity of the points reflected from surrounding objects. We show how different layers of the intensity map can be used to accurately differentiate solid obstacles from passable objects, and identify transparent obstacles. Finally, we detail our obstacle inflation strategy, which enables robots to navigate through passable obstacles

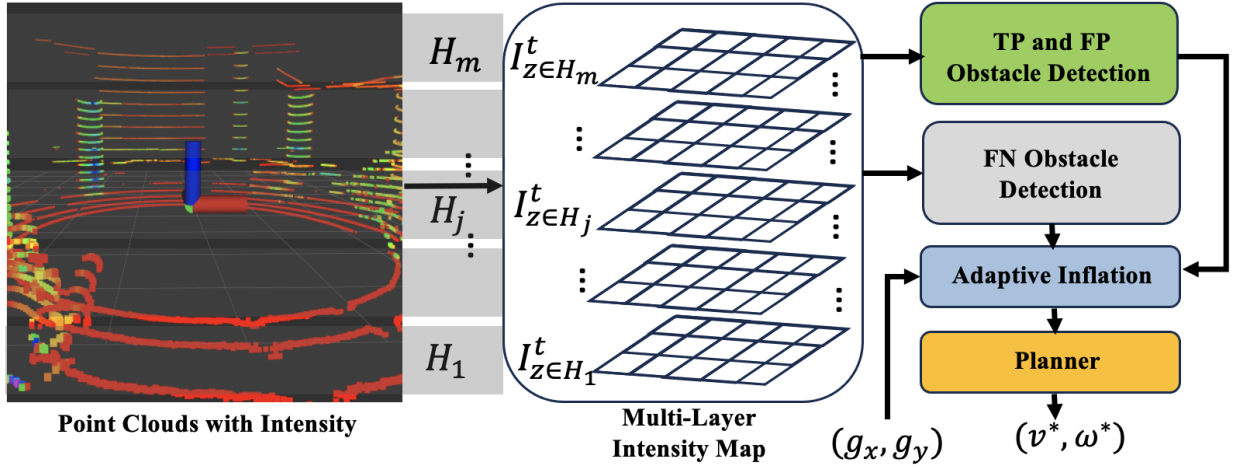


Figure 5.3: MIM's overall system architecture. At time instant t , the intensities of reflected point clouds from a height interval H_j (grey rectangles) are used to construct a 2D grid map layer $I_{z \in H_j}^t$ according to equation 5.1. Several of such layers are stacked to form an MIM. Certain layers from MIMs can be used to detect TP, FP, and FN obstacles. The adaptive inflation enlarges the truly solid (TP, FN) obstacles that the robot must avoid in the direction opposite to the robot's goal (g_x, g_y) ' direction, and ignores passable obstacles (FP). The planner finally uses the inflated map to compute collision-free linear and angular velocities v^*, ω^* .

and narrow passages. Fig. 5.3 shows our overall proposed architecture.

5.3.2.1 Constructing the Multi-Layer Intensity Map

We construct a single layer of the intensity map corresponding to a height interval H_j at any time instant t as follows,

$$I_{z \in H_j}^t(r, c) = \frac{\sum_x \sum_y int}{g^2}$$

$$\forall x \in [x_{low}, x_{low} + g], y \in [y_{low}, y_{low} + g],$$

$$x_{low} = *(r - \frac{n}{2}) \cdot g \text{ and } y_{low} = *(c - \frac{n}{2}) \cdot g$$

$$H_j = [low_j, high_j], low_j \leq high_j,$$
(5.1)

where $*$ represents the floor operation, low_j , and $high_j$ are the limits for all the points whose intensities must be considered for the map along the z direction. Extending this definition, we

construct a *multi-layer* intensity map as a stacking of m layers as,

$$I_{ML}^t(r, c) = [I_{z \in H_1}^t(r, c) \mid \dots \mid I_{z \in H_j}^t(r, c) \mid \dots \mid I_{z \in H_m}^t(r, c)], \quad (5.2)$$

where $H_1, \dots, H_j, \dots, H_m$ are non-overlapping height intervals. We choose to stack multiple non-overlapping layers at various heights instead of combining the points' intensities at all heights into a unified layer. This is due to the flexibility that multiple layers provide in analyzing and modifying them individually. Furthermore, individual layers can be combined after modification and used for planning the robot's trajectories. We highlight these benefits in the following sections.

5.3.2.2 Obstacle Detection

In this section, we describe how challenging obstacles such as tall grass (FP), string/beaded curtains (FP), and transparent objects (FN) can be detected using our multi-layer intensity map.

Differentiating True and False Positive Obstacles: Existing methods that use various sensor modalities typically detect many thin, pliable obstacles such as tall grass, and passable objects such as string/beaded curtains as solid obstacles that must be avoided. During navigation, such inaccurate detections cause the robot to freeze or oscillate perpetually without reaching its goal. We show how such false positive obstacles can be detected using the multi-layer intensity map.

Let us consider three layers $I_{z=0}^t, I_{z \in (0, h]}^t, I_{z \in [-h, 0)}^t$ of the intensity map. If a grid location (r, c) belonging to all three layers satisfies the following condition $\mathcal{C}$, we classify that grid as a false positive obstacle:

$$\mathcal{C}(r, c) : I_{z=0}^t(r, c), I_{z \in (0, h]}^t(r, c), I_{z \in [-h, 0)}^t(r, c) \leq \Gamma. \quad (5.3)$$

Here, Γ is an intensity threshold. Using this condition, we construct TP and FP intensity maps for planning as,

$$I_{TP}^t(r, c) = \max(I_{z=0}^t(), I_{z \in (0, h]}^t(), I_{z \in [-h, 0)}^t()) \quad (5.4)$$

$$\forall (r, c) \mid \mathcal{C}(r, c) \text{ is False}$$

$$I_{FP}^t(r, c) = \max(I_{z=0}^t(), I_{z \in (0, h]}^t(), I_{z \in [-h, 0)}^t()) \quad (5.5)$$

$$\forall (r, c) \mid \mathcal{C}(r, c) \text{ is True}$$

Detecting False Negative Obstacles: False negative obstacles are typically transparent objects that allow most of the laser rays from a sensor to pass through. However, a small neighborhood of points with low intensities is detected for laser rays that are incident $\sim 0^\circ$ on a transparent surface only along the $z = 0$ plane [278]. Our approach extrapolates this small neighborhood to detect solid transparent obstacles.

Consider two layers $I_{z=0}^t$ and $I_{z=-\epsilon}^t$ in the multi-layer intensity map at time instant t . Here, ϵ is a small positive value. To isolate the low-intensity neighborhood of points reflected from the transparent object, we first calculate the element-wise difference between the two layers $I_{z=0}^t \ominus I_{z=-\epsilon}^t$. This removes all the points reflected from the same obstacles in both layers and retains only the points corresponding to the small glass neighborhood. To indicate the presence of transparent obstacles for subsequent time steps, we transform $I_{z=0}^t \ominus I_{z=-\epsilon}^t$ based on the robot's motion as,

$$I_{FN}^t = T \cdot (I_{z=0}^t \ominus I_{z=-\epsilon}^t), \quad (5.6)$$

where T is a 4×4 transformation matrix whose rotational component is based on the robot's yaw,

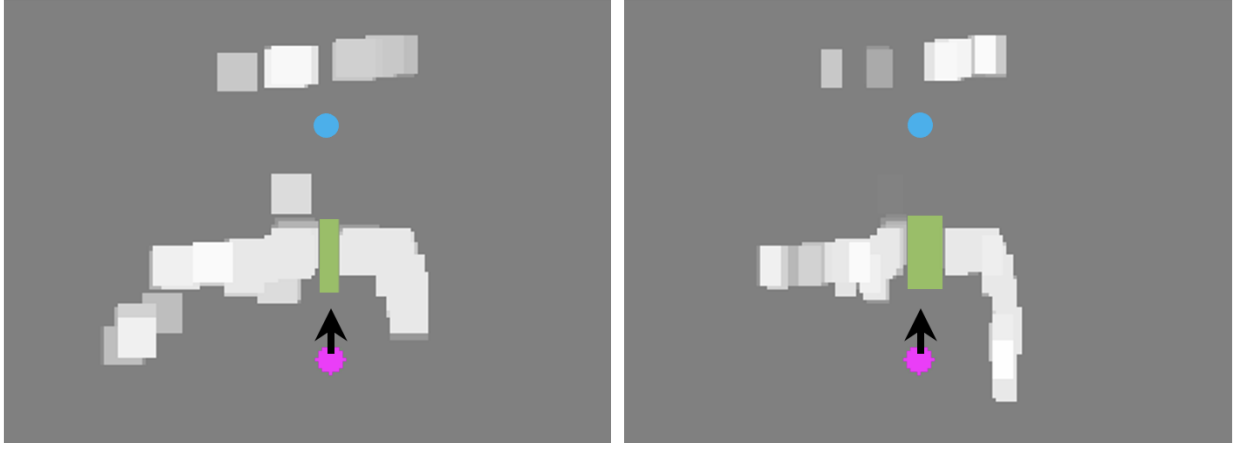


Figure 5.4: The robot’s position and its goal are denoted by the pink and blue circles respectively. The robot’s heading and goal direction coincide in this case. **[Left]**: Uniformly inflating obstacles using an $e \times e$ kernel with all ones near a narrow passage closing up the available free space (green rectangle). **[Right]**: Adaptively inflating the obstacles based on the robot’s goal direction preserves the free space while inflating the obstacles in the direction opposite to the robot’s goal direction.

and the translational component is based on its motion from time t to $t + 1$. Finally, we augment subsequent time’s I_{FN}^{t+1} using I_{FN}^t as,

$$I_{FN}^{t+1} = I_{FN}^{t+1} \cup I_{FN}^t. \quad (5.7)$$

5.3.2.3 Adaptive Obstacle Inflation

In narrow and dense scenarios, uniformly inflating obstacles (as explained in section 5.3.1.4) could close the available free space (see fig. 5.4) leading to the robot freezing problem [54]. If obstacles are not inflated, the robot could move close to the obstacles and even collide with them. Additionally, false positive obstacles that can be traversed need not be inflated. Therefore, our approach adaptively inflates the obstacles based on the robot’s goal direction. Obstacles are majorly inflated in the direction opposite to the robot’s goal/preferred direction and minorly inflated in all other directions. This ensures that the robot does not navigate too close to a solid obstacle, while also never closing free space near the narrow passages.

Our inflation is performed using the convolution operation on obstacles by computing the appropriate kernel matrices K of size $e \times e$ as follows. Let (g_x, g_y) be the goal location relative to the robot's coordinate frame. The goal direction can be defined by the slope $\tan \frac{g_y}{g_x}$. To design a kernel matrix K to inflate obstacles in the opposite direction, we first find the line along the goal direction relative to the kernel, passing through its center $(\frac{e}{2}, \frac{e}{2})$ as,

$$\begin{aligned} f(r^K, c^K) : c^K - \tan(g_y/g_x) \cdot r^K + \text{const} &= 0 \\ \text{const} &= \frac{e}{2}(\tan(g_y/g_x) - 1). \end{aligned} \quad (5.8)$$

Here, (r^K, c^K) represent the row and column on the kernel. Next, the kernel can be constructed as,

$$K(r^K, c^K) = \begin{cases} 1 & \forall \{r^K, c^K | f(r^K + j, c^K + j) = 0\} \\ 0 & \text{Otherwise.} \end{cases} \quad (5.9)$$

Here, $j \in [-\text{padding}, \text{padding}]$ is added to the row and column indices to control the level of thickness to inflate an obstacle. Finally, using K , we convolve our multi-layer intensity map as,

$$I_{in\,flate}^t = (I_{TP}^t \cup I_{FN}^t) \circledast K. \quad (5.10)$$

$I_{in\,flate}^t$ contains inflated True Positive and False Negative obstacles. To add information about false positive obstacles prior to planning, we perform,

$$I_{plan}^t = I_{in\,flate}^t \cup I_{FP}^t, \quad (5.11)$$

where I_{plan}^t is a 2D grid map containing TP, FP, FN, and free space represented by various grids, which can be used as a cost map for motion planning.

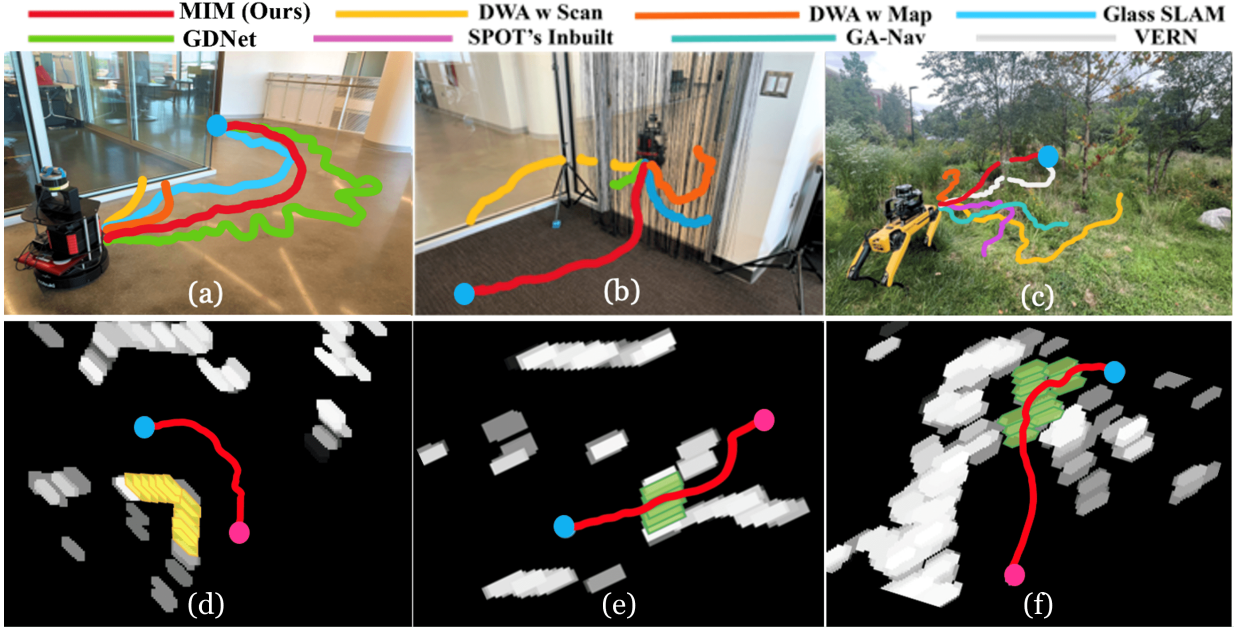


Figure 5.5: Robot trajectories when navigating in different complex indoor and outdoor environments using various methods. (a) MIM identifies transparent objects such as glass in real-time and avoids it, (b) MIM identifies string curtains as safe, passable obstacles while other methods detect them as solid, and impassable. (c) MIM accurately identifies pliable tall grass regions (which register lower intensities on I_{plan}^t) to navigate through, avoiding trees. (d,e,f) show the corresponding I_{plan}^t for the scenarios above. White, grey, and black colors represent intensities in a decreasing order of magnitude. The robot’s starting location is in pink, and the goal is represented in blue. The yellow in (d) represents the glass (FN obstacle) extrapolated in real-time by our method in section 5.3.2.2. Green in (e, f) represents passable, non-solid obstacles such as curtains and tall grass respectively. MIM-based navigation’s trajectory is overlayed for reference. (d, e, f) also depict obstacles inflated based on the robot’s goal direction.

5.3.2.4 Integration with Planning Methods

The final planning intensity map I_{plan}^t can be used with any motion planner as a cost map to evaluate a candidate linear and angular velocity’s (v, ω) obstacle or collision cost. This can be computed by extrapolating the trajectory produced by (v, ω) relative to I_{plan}^t as in [251, 267] as,

$$\begin{aligned}
 traj^{I_{plan}^t} &= [(r_1, c_1), \dots, (r_j, c_j), \dots, (r_p, c_p)] \\
 \text{Obstacle Cost} &= \sum_{j=1}^p I_{plan}^t(r_j, c_j).
 \end{aligned} \tag{5.12}$$

Additionally, I_{plan}^t can be used as an observation with deep reinforcement learning-based

navigation methods [27] to improve the trained model’s understanding of the solidity of surrounding obstacles.

5.3.3 Results and Evaluations

In this section, we summarize our method’s implementation details and evaluation metrics. Then, we present the details of the experiments conducted to highlight the benefits of our approach.

5.3.3.1 Implementation

We use a Boston Dynamics Spot robot for outdoor experiments and a Turtlebot 2 robot for indoor experiments. The Spot robot is equipped with an Intel NUC 11 with Intel i7 CPU and NVIDIA RTX 2060 GPU. The Turtlebot is equipped with a laptop with an Intel i9 CPU and an Nvidia RTX 2080 GPU. Both robots are equipped and a Velodyne VLP16 LiDAR. We particularly use Velodyne LiDAR’s dual return mode since it captures low intensities in the point cloud better [278]. We use $n = 200$, and $m = 4$ layers in our implementation (one for $I_{z \in [-h, 0]}^t, I_{z \in (0, h]}^t, I_{z=0}^t, I_{z=-\epsilon}^t$). The height ranges are adjusted based on the robot’s height.

5.3.3.2 Comparison Methods and Evaluation Metrics

We compare the improvements in navigation while using MIMs with various existing methods. We use MIMs with the dynamic window approach (DWA) [177] and calculate candidate velocity’s obstacle costs as described in section 5.3.2.4. For a fair comparison, we use DWA as the baseline planner and integrate other methods for obstacle perception with it for navigation. We compare with various indoor and outdoor navigation methods that use a variety of perception

inputs such as RGB/depth image, 2D laser scans, or occupancy grids created by any proximity sensor.

DWA is a local planner that can use a 2D LiDAR scan [177] or an occupancy map [230] to perform obstacle avoidance. Spot's in-built autonomy incorporates a set of stereo cameras around the robot to estimate the obstacles and ground plane to navigate to a goal. VERN is a vegetation-aware navigation approach that uses an RGB image-based vegetation classifier and a set of 2D occupancy grid maps for perception. In indoor scenarios, Glass-SLAM [279] uses the specular reflection of laser beams from the glass to map environments that include glass. GDNet [16] is a semantic segmentation method that uses RGB images to segment glass from a scene. We also perform ablations studies between using adaptive and uniform inflation using MIMs.

Success Rate - Number of successful goal-reaching attempts (without collisions with solid objects or freezing behaviors) out of the total number of trials.

Normalized Traj. Length - The ratio between the robot's trajectory length and the straight-line distance to the goal from the starting location averaged over all runs.

F-Score - A measure of object detection accuracy calculated as a weighted average of the precision and recall. Values are between 0 and 1, where 1 indicates the best accuracy. We use human detection of TP, FP, and FN obstacles to calculate precision and recall.

Inference Time - Time taken from the instant an obstacle is viewed to the instant when it is detected.

5.3.3.3 Testing Scenarios

- **Scenario 1** - Indoor scenario with glass, concrete walls, and pillars (Fig. 5.5a).

Scenario	Method	Sucess Rate (%)	Norm. Traj. Length	F-Score
Scn. 1	DWA with laser scan [177]	0	0.342	0.12
	DWA with occupancy map [230]	0	0.415	0.20
	Glass-SLAM [279]	30	0.895	0.76
	GDNet [16]	50	0.633	0.69
	MIM (Uniform)	80	1.256	0.82
	MIM (Adaptive)	80	1.052	0.82
Scn. 2	DWA with laser scan [177]	0	0.765	0.17
	DWA with occupancy map [230]	0	0.355	0.16
	Glass-SLAM [279]	0	0.421	0.46
	GDNet [16]	0	0.318	0.65
	MIM (Uniform)	30	1.134	0.79
	MIM (Adaptive)	70	1.141	0.79
Scn. 3	DWA with laser scan [177]	20	1.571	0.35
	DWA with occupancy map [230]	20	1.482	0.33
	Spot's Inbuilt Autonomy	10	0.311	0.38
	VERN [251]	70	1.154	0.75
	MIM (Uniform)	40	1.292	0.84
	MIM (Adaptive)	80	1.072	0.84
Scn. 4	DWA with laser scan [177]	0	1.543	0.37
	DWA with occupancy map [230]	0	1.412	0.36
	Spot's Inbuilt Autonomy	0	0.298	0.31
	VERN [251]	50	1.267	0.68
	MIM (Uniform)	60	1.248	0.74
	MIM (Adaptive)	70	1.146	0.74

Table 5.1: Performance comparison between using MIMs for navigation versus other methods in indoor and outdoor scenarios using various metrics. We observe that MIMs are versatile in detecting a variety of perceptually challenging obstacles and aiding navigation.

- **Scenario 2** - Indoor scenario with a narrow passage covered with a beaded/string curtain (Fig. 5.5b).
- **Scenario 3** - Outdoor scenario with tall grass, bushes, and trees separated from each other (Fig. 5.5c).
- **Scenario 4** - Outdoor scenario with tall grass, bushes, and trees closely intertwined with each other creating narrow passages (Fig. 5.1).

Method	Inference Time (sec)	Training Time
MIM	0.020 on CPU	NA
Glass-SLAM	4.5 on CPU	NA
GNet	0.112 on GPU	12-15 hours
VERN	0.083 on GPU	6-8 hours

Table 5.2: The inference rates and training time (where applicable) for several methods that detect glass (Glass-SLAM [279], GNet [16]), and pliable vegetation (VERN [251]). MIMs are capable of detecting transparent obstacles such as glass, and passable tall grass in real-time, faster than existing methods that use LiDARs and RGB images.

5.3.3.4 Analysis and Comparison

We present the qualitative navigation experiment results for the four scenarios in Fig. 5.5 and the quantitative results in Table 5.1. Scenarios 1 and 2 are complex indoor settings, whereas scenarios 3 and 4 are outdoor settings. We observe that MIM demonstrates the highest success rate compared to the other methods in all four scenarios.

In scenario 1, 2D laser scan-based and occupancy map-based DWA planners fail to identify the glass region since they do not incorporate LiDAR intensity for object detection. GNet-based indoor segmentation fails to identify the glass region from RGB inputs consistently due to lighting changes, and strong lights reflected from the floor. Hence, these methods lead to collisions with the glass by assuming it to be free space. The detecting glass in SLAM can identify the glass region using LiDAR intensity. However, it does not avoid glass all the time since the glass is constructed slower than the robot’s motion. MIM’s multi-layer map formulation leads to consistent glass detection which also results in a higher F-score compared to the other methods.

Scenario 2 includes a passable string curtain which is detected as an obstacle from the 2D laser scan and occupancy map-based DWA methods. Hence, these methods attempt to avoid the

curtain and collide with the glass during navigation. Further, the glass detection SLAM method identifies both the curtain and the glass as obstacles. Hence, all three methods demonstrate poor success rates in scenario 2. GDNet identified both the open door and the glass near it as glass preventing the robot from entering through the door. In contrast, our MIM demonstrates a significantly high success rate irrespective of the lighting conditions due to the 360° LiDAR-based intensity map formulation.

Scenarios 3 and 4 include traversable tall grass and non-traversable bushes and trees. However, the two DWA methods identify all such vegetation regions as obstacles due to 2D laser scan-based obstacle detection causing freezing behaviors and longer detours during navigation. Similarly, Spot’s inbuilt autonomy struggles to estimate the ground and free space from its vision-based perception in both Scenario 3 and Scenario 4. Hence, the robot demonstrates highly unstable motion in vegetation. In scenario 3, VERN is able to navigate through tall grass while avoiding trees and bushes using its vision-based classifier and the occupancy map formulation. However, VERN’s vision-based classifier could not detect trees behind the grass region in scenario 4 since they are closely intertwined. In contrast, MIM’s multi-layer intensity map representation identifies such hidden solid objects to avoid during navigation. Hence, MIM demonstrates a relatively higher success rate and F-score in scenario 4. In all scenarios, MIMs (uniform and adaptive) have the highest F-score. The inaccuracies in detecting obstacles in MIMs occur when the robot/LiDAR is closer than 0.5 meters away from an obstacle, whereas 3D LiDARs typically have a blindspot.

Benefits of Adaptive Inflation: We use scenarios 2 and 3, which contain narrow passages to highlight the benefits of our adaptive inflation formulation. Laserscan and occupancy map-based DWA, and MIM (Uniform) use uniform inflation around the obstacles to avoid collisions.

This closes the narrow passages and represents them as obstacle regions, resulting in freezing or longer trajectories in the presence of narrow passages between the obstacles. However, our adaptive inflation preserves the narrow free spaces in the cost map while inflating the obstacles. Hence, MIM can navigate through such spaces (e.g., through a small door in scenario 2 and between the trees in scenario 3) and reach the goals using shorter trajectories.

Inference Time: We compared the inference times (Table 5.2) of using MIMs to detect glass and pliable vegetation with other methods that either detect glass (Glass-SLAM, GDNet) or vegetation (VERN) on the Intel NUC described in section 5.3.3.1. We observe that apart from being versatile in detecting obstacles, MIMs are computationally light to be used with a robot’s limited onboard computing resources. GDNet and VERN use RGB images passed through deep neural networks and require extensive prior training. While Glass-SLAM does not need training, it requires ~ 4.5 seconds to update a map with obstacles and multiple runs to detect glass.

5.3.4 Limitations

Since our multi-layer map representation is based on point cloud intensity, it cannot identify passable objects such as cloth curtains and metal fences. This is especially important because the navigability of such objects depends on the context (e.g., window curtains may not be passable but door curtains are generally passable). Hence, a semantic understanding of the environment is required for such cases. Further, our method cannot detect extremely thin objects such as thin poles since a 3D point cloud may not be able to capture sufficient number of samples.

5.4 CoNVOI

In this section, we explain the background definitions and assumptions (section 5.4.1) used in formulating CoNVOI, its methodology (section 5.4.2), evaluations (section 5.4.3), and limitations (section 5.4.4).

5.4.1 Background

In this section, we introduce and define the symbols and key concepts used in our work. Apart from these concepts, CoNVOI’s local planner that follows the reference path from the VLM is extended from DWA (section 4.3.1.2).

5.4.1.1 Symbols, Definitions, and Assumptions

CoNVOI addresses the challenge “what navigation behaviors should a robot exhibit in its current local environmental context”. This involves determining the locations within the robot’s field of view to navigate toward to exhibit these behaviors, extending beyond simple obstacle avoidance.

For our formulation, we assume a robot equipped with an RGB camera and a 2D or 3D LiDAR rigidly attached to a wheeled or legged mobile robot base. The camera provides an RGB image I_t^{RGB} to view the environment and the LiDAR provides 2D laser scans at time instant t . Our formulation uses an $n \times n$ robot-centric occupancy grid map $\mathcal{O}_t$ obtained using the LiDAR’s laser scan to represent the environmental obstacles. The grids of $\mathcal{O}_t$ are marked with 1’s at grids that contain obstacles, and 0’s otherwise. The obstacles are inflated by the robot’s radius, and the robot is considered as a point at $(n/2, n/2)$, and the set of obstacles is denoted as obs_t . The pixels

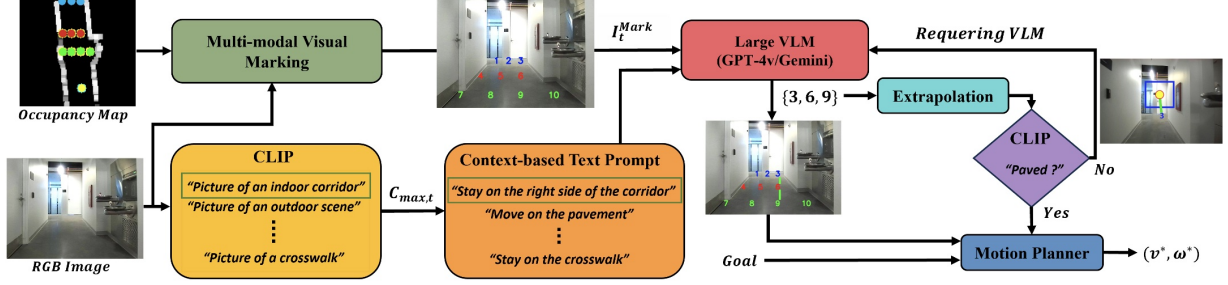


Figure 5.6: CoNVOI’s architecture utilizes CLIP to interpret the context of the robot’s environment from an RGB image (I_t^{RGB}), identifying features such as indoor corridors, social scenarios with people, outdoor terrains, etc. Next, CoNVOI queries a large VLM with a context-based text prompt, and the RGB image marked with numbers (I_t^{Mark}) in the free space detected in an occupancy grid map to generate a reference path (in green) that adheres to explicit and implicit social rules (e.g., staying on pavement, using crosswalks). A dedicated motion planner then follows this path while avoiding obstacles. To prevent unnecessarily requering the large VLM, we extrapolate this reference path linearly and check if the extrapolated point (in yellow) lies on a paved path (sidewalk, corridor, etc) to either use it for navigation, or requery the VLM again. Instead of using VLMs for direct robot control or for well-addressed tasks such as goal-reaching and obstacle avoidance, we leverage its context-understanding capabilities to achieve more intricate, zero-shot navigation behaviors using a separate planner.

in I_t^{RGB} are correlated with the grids in $\mathcal{O}_t$ using a homography perspective projection.

The robot is controlled using linear and angular velocity commands (v, ω) , respectively.

We consider *three coordinate frames*:

1. robot - a frame attached to the robot’s center of mass with X, Y, and Z axes pointing forward, leftward, and upward relative to the robot, 2. Occupancy map ($\mathcal{O}$) - a frame attached at the center of $\mathcal{O}_t$ with X, and Y axes pointing forward and leftward, and 3. RGB - a frame attached to the top-left corner of the RGB image obtained from the camera, with X, and Y axes pointing rightward (across columns), and downward (across rows). Quantities belong to the frame indicated by their superscripts (e.g. x^{rob} belongs to the robot frame). Transformation matrices between these frames are represented as T_{rob}^{odom} , $T_{\mathcal{O}}^{rob}$, $T_{RGB}^{\mathcal{O}}$. These matrices transform points in the subscript frame to the superscript frame. Finally, we use symbols i, j to denote indices.

CoNVOI’s overall architecture is shown in Fig. 5.6.

5.4.2 CoNVOI: Context-aware Navigation using VLMs in Outdoors and Indoors

CoNVOI uses two novel components: 1. a context-based prompting method to query a large VLM with an appropriate *behavior* text prompt, and 2. a multi-modal visual marking method to prepare the large VLM’s input image. Using both components CoNVOI extracts a reference trajectory $\mathcal{T}_{ref}$ from the VLM. $\mathcal{T}_{ref}$ obeys the explicit and implicit rules associated with the robot’s current context/scenario, and also encodes human-like preferences (e.g. opting to walk on pavement instead of grass) during navigation.

5.4.2.1 Context-based Behavior Prompt

VLMs utilize an image and text prompt inputs to output a text response. CoNVOI’s first component is an approach to frame a context-based behavioral text prompt for the VLM. To this end, we formulate a set of contexts/scenarios $\mathcal{C} = \{c_1, c_2, \dots, c_k\}$ that the robot would operate in (e.g. indoor corridors, in the presence of humans, outdoor terrains, crosswalk, etc.), and a set of robot behaviors for each context. The behaviors $\mathcal{B} = \{b_1, b_2, \dots, b_k\}$ are specified as simple text phrases such as “keep close to the right wall” or “move on pavement”, etc. Our formulation allows for easily adding new scenarios and associated behaviors to $\mathcal{C}$ and $\mathcal{B}$, respectively.

Next, we query a light-weight VLM (e.g. CLIP [242]) with I_t^{RGB} and text prompts T_i framed as “*This is a picture of c_i* ” ($\forall i$), and obtain a set of probabilities $\{p_1, p_2, \dots, p_k\}$ ($\sum_i p_i = 1$), where $p_i = P(c_i | I_t^{RGB}, \{T_1, T_2, \dots, T_k\})$ is the probability that the context/scenario in I_t^{RGB} corresponds to context c_i . Let $c_{max,t}$ correspond to the context with maximum probability at time t , and b_{max} be its associated robot behavior. Next, we frame a behavioral prompt $T_{b_{max}}$ in the format “*You are navigating a robot in this scenario. The numbers marked in the image denote*

regions where you can navigate the robot. Pick a list of numbers marked in the image such that the robot _____”, and the blank is filled with the behavioral phrase in b_{max} .

5.4.2.2 Multi-modal Visual Marking

To prepare the image input for the VLM, we take into account the following considerations. Firstly, VLMs do not reliably infer spatial relationships, such as the distance of a point from the camera and the relative locations of various regions in the image (e.g., *point A is closer to the right side of the image*) [280]. Secondly, the VLM does not need to *focus* on obstacle regions since its task is solely to identify regions within the free space to identify *how* the robot should navigate. To this end, we propose to use the correlation between the robot’s occupancy grid map $\mathcal{O}_t$ and the RGB image I_t^{RGB} to detect the free space in the robot’s vicinity, and overlay markers (numbers) [245] on the corresponding free space in I_t^{RGB} . This approach combines the benefits of the spatial information encoded in occupancy grids with the semantic information in RGB images and results in a richer input to the VLM.

Detecting Free Regions to Add Markers: We consider l rows and m columns of grids each on $\mathcal{O}_t$ as candidate locations where the robot could traverse to exhibit context-aware behaviors. These grids are uniformly spaced by a distance d_{col} along the columns, and d_{row} along the rows. Let us consider the ij^{th} candidate grid $\mathbf{g}_{ij}$, $i \in \{1, 2, \dots, l\}, j \in \{1, 2, \dots, m\}$. Let L_{ij} be the line connecting $\mathbf{g}_{ij}$ and the robot’s location $(n/2, n/2)$ in $\mathcal{O}_t$. $\mathbf{g}_{ij}$ is considered as a navigable candidate if,

$$L_{ij} \cap obs_t = \emptyset. \quad (5.13)$$

This ensures that locations obstructed by obstacles from the robot and its camera's view are not considered viable for context-aware navigation. We construct the set of candidate coordinates satisfying equation 5.13 as $X_t^{\mathcal{O}} = \{\mathbf{g}_{ij}\} \forall (i, j)$. The points in $X_t^{\mathcal{O}}$ are transformed into the RGB frame as,

$$(x_{ij}^{RGB}, y_{ij}^{RGB}) = T_O^{RGB} \cdot \mathbf{g}_{ij} \quad \forall \mathbf{g}_{ij} \in X_t^{\mathcal{O}}. \quad (5.14)$$

We construct the set of candidate coordinates as $X_t^{RGB} = \{(x_{ij}^{RGB}, y_{ij}^{RGB})\} \forall (i, j)$. The points in X_t^{RGB} are ordered from left to right and row by row (see Fig. 5.6). Next, we mark numbers on all the points in X_t^{RGB} in ascending order on I_t^{RGB} and refer to this marked image as I_t^{Mark} .

5.4.2.3 Prompting the Large VLM

Finally, using $T_{b_{max}}$ and I_t^{Mark} , we prompt a large VLM [237, 240] to extract a reference path as,

$$\mathcal{M}_{ref} = \text{VLM}(I_t^{Mark}, T_{b_{max}}). \quad (5.15)$$

Here, $\mathcal{M}_{ref}$ is a list of numbers (markers) marked on I_t^{Mark} . Their corresponding set of grid locations relative to $\mathcal{O}_t$ are obtained by using the numbers in $\mathcal{M}_{ref}$ as indices since they are marked orderly in I_t^{Mark} . Hence, the reference path in the occupancy map frame is $\mathcal{T}_{ref}^{\mathcal{O}} = X_t^{\mathcal{O}}(\mathcal{M}_{ref})$. Their locations in the robot frame can be calculated as $\mathcal{T}_{ref}^{rob} = T_O^{rob} \cdot \mathcal{T}_{ref}^{\mathcal{O}}$.

5.4.2.4 Reference Path Following

When the robot follows the reference path $\mathcal{T}_{ref}^{rob} = \{(x_{ref,1}^{rob}, y_{ref,1}^{rob}), (x_{ref,2}^{rob}, y_{ref,2}^{rob}), \dots, (x_{ref,last}^{rob}, y_{ref,last}^{rob})\}$ $i = \{1, 2, \dots, last\}$, it exhibits the most appropriate navigation behaviors for the context it encounters. These coordinates are repeatedly transformed into the robot's coordinate frame as the robot moves. To follow the reference path, we modify the motion planner [29] explained in section 4.3.1.2 as follows. Let θ_{goal} be the angle between the robot's current heading direction and the vector pointing to its goal direction. We first formulate a reference path following cost as,

$$ref(v, \omega) = |y_{ref,i}^{rob} - y_{t+t_{hor}}^{rob}|, \quad y_{t+t_{hor}}^{rob} \in traj^{rob}(v, \omega),$$

$$\text{and } i \text{ s.t } x_{ref,i}^{rob} > 0 \text{ and} \quad (5.16)$$

$$x_{ref,i}^{rob} = \min(x_{ref,1}^{rob}, x_{ref,2}^{rob}, \dots, x_{ref,last}^{rob}),$$

Equation 5.16 ensures that the robot picks (v, ω) trajectories that minimize the lateral distance between the robot and a reference path point that is immediately ahead of the robot. A new objective function for planning as,

$$Q_2(v, \omega) = \begin{cases} Q_1(.) + \alpha_4 \cdot ref(.) & \text{if } |\theta_{goal}| \leq \theta_{fov}, \\ Q_1(.) & \text{Otherwise.} \end{cases} \quad (5.17)$$

Here, $Q_1(.)$ is DWA's objective function (equation 4.1). θ_{fov} is the field of view of the robot's camera. This condition helps to ensure that the robot does not stray away from the goal direction by following the VLM's reference paths.

5.4.2.5 Reference Path Extrapolation and Re-querying

Since large VLMs require several seconds to respond to queries (such as in equation 5.15), continuously querying to obtain new reference paths is computationally infeasible for real-time navigation. Therefore, CoNVOI reduces the frequency of querying the large VLM by extrapolating the current reference path, if the context has not changed.

Once the robot obtains a reference path and starts following it, CoNVOI repeatedly queries CLIP to check for context changes. If $c_{max,t} = c_{max,t'} (t' > t)$, and $\text{dist}((x_{ref,last}^{rob}, y_{ref,last}^{rob})) < d_{thresh}$ (since $(0, 0)$ is the robot's location in its own frame, and d_{thresh} is a distance threshold), we linearly extrapolate the current reference path by fitting a line to the points in $\mathcal{T}_{ref}^{rob}$ and extending it further to a point $(x_{ref,e}^{rob}, y_{ref,e}^{rob})$ by distance d_{row} . Next, we check if the point $(x_{ref,e}^O, y_{ref,e}^O)$ is obstacle-free (similar to section 5.4.2.2). If it is, we add the point to the RGB image at $(x_{ref,e}^{RGB}, y_{ref,e}^{RGB})$, and crop a $n_{pat} \times n_{pat}$ image patch I^{pat} (see Fig. 5.6) to query CLIP if the patch is an image of a paved (indoors or sidewalks) or unpaved (grass, gravel, asphalt) surface. If $CLIP(I^{pat})$ is paved, $(x_{ref,e}^{rob}, y_{ref,e}^{rob})$ is used as the next waypoint for the planner to follow. Otherwise, the large VLM is queried for the next reference path.

If the context has changed, CoNVOI queries the large VLM once $\text{dist}((x_{ref,last}^{rob}, y_{ref,last}^{rob})) < d_{thresh}$. d_{thresh} is chosen such that $d_{thresh} \approx v_{max} \cdot t_{query}$, where v_{max} is the maximum linear velocity of the robot and t_{query} is the average time for a large VLM to respond to a query.

5.4.3 Results and Evaluations

In this section, we explain CoNVOI's implementation on real robots, and analyze and evaluate its performance through comparisons and ablations.

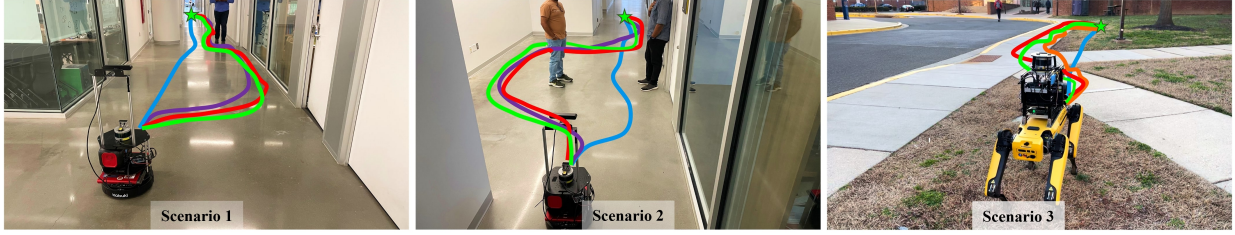


Figure 5.7: Robot trajectories when navigating in different complex indoor and outdoor environments using various methods: CoNVOI (in green), teleoperated by a human (in red), DWA [29] (in blue), Frozone [54] (in violet), GA-Nav [18] (in orange). CoNVOI exhibits social-compliant behaviors such as not moving in-between humans even if there is sufficient space, similar to Frozone, a method formulated for indoor social navigation. CoNVOI’s behaviors also match GA-Nav, a semantic segmentation-based navigation approach that prefers to navigate on smooth, well-paved outdoor terrains. CoNVOI achieves these behaviors in a zero-shot manner, and does not require domain-specific fine-tuning. CoNVOI’s trajectories also closely match human-teleoperated ground truth paths (in red).

5.4.3.1 Implementation

CoNVOI is implemented on a Turtlebot 2 for indoor evaluation, and on a Boston Dynamics Spot robot for outdoor evaluation. Both robots are equipped with a Velodyne VLP16 LiDAR, and a Zed 2i camera. CoNVOI is executed on a laptop with an Intel i7 11th generation CPU and an Nvidia RTX 3060 GPU. For context detection, CLIP is executed locally on the laptop, and for the reference path queries, we use the online API for GPT-4v [237], and Gemini [240]. We use the following values for the parameters in our formulation: $n = 200$, $\alpha_1 = 10$, $\alpha_2 = 7$, $\alpha_3 = 1$, $\alpha_4 = 7.5$, $l = 2$ for outdoors, 3 for indoors, $m = 6$, $d_{row} = 2.5m$, $d_{col} = 2m$, $k = 5$, $v_{max} = 0.3m/s$, $t_{query} = 5s$, $d_{thresh} = 4m$, $n_{pat} = 200$.

5.4.3.2 Comparison Methods

We compare CoNVOI with several existing methods for indoor and outdoor navigation. They are:

- **DWA** [29]: The dynamic window approach, a baseline motion planner that performs simple

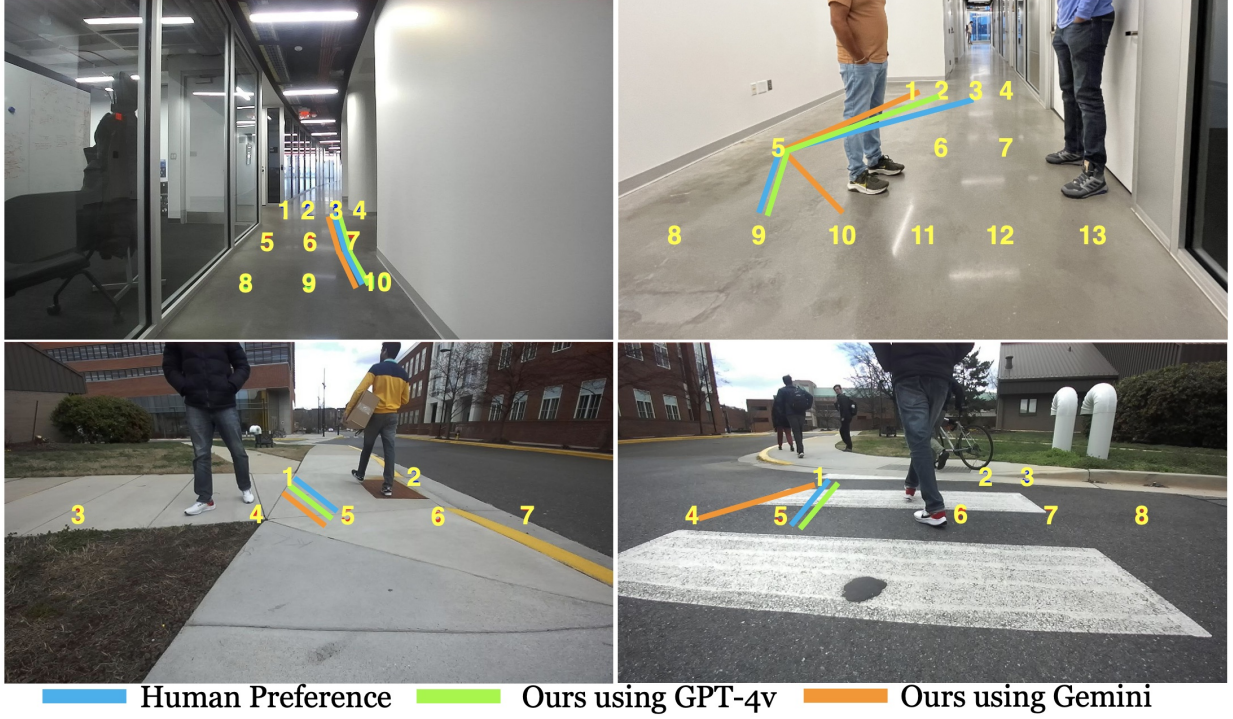


Figure 5.8: Qualitative comparison of the reference trajectory generated by GPT-4v [237] (in green), and Gemini [240] (in orange) compared with human provided ground truth (in blue) by connecting the marked numbers (in yellow). We observe that the reference paths generated by the VLMs is comparable to that of the human-preferred path in many complex indoor and outdoor environments.

collision avoidance and goal-reaching behaviors.

- **GA-Nav** [18]: A planner that uses semantic segmentation-based outdoor terrain classification to assign traversability costs to various terrains and computes trajectories on smooth (low-cost) terrains.
- **PSP-Net** [219]: A planner similar to GA-Nav but employing PSP-Net for semantic segmentation.
- **Frozone** [54]: A social-compliant indoor navigation method that computes trajectories that are least obtrusive to surrounding pedestrians. We enhance Frozone’s formulation by incorporating a method to detect corridors from the robot’s local occupancy grid. Additionally, we assign higher costs to regions closer to the left wall, which encourages the robot to stay

closer to the right side.

In addition to baseline methods, we conduct thorough ablation studies by analyzing our approach’s reference path selection performance by removing its multi-modal visual marking, and context-based prompting components.

5.4.3.3 Evaluation Metrics

We use the following quantitative metrics to evaluate the navigation performance of all the methods in real-world scenarios.

- **Fréchet Distance w.r.t. Human Teleoperation:** Measures the Fréchet distance [281] (a measure of similarity between two curves) between a human teleoperated robot trajectory versus a comparison method’s trajectory.
- **Normalized Trajectory Length:** The robot’s trajectory length normalized over the straight-line distance to the goal.
- **Mean Velocity:** The robot’s linear velocity (that indicates its progress towards the goal) averaged over all the trials.

We use the following metrics to evaluate CoNVOI’s reference path selecting capabilities by comparing it with a human chosen ground truth points in the marked image.

- **Reference Path Error:** Measures the distance between the mean points in the ground truth path and the VLM’s output reference path.
- **Cosine Similarity:** Quantifies the alignment between the ground truth and the VLM’s reference path in terms of orientation.

- **Number of unacceptable paths:** Counts the number of times a reference path was on an unacceptable terrain/location (e.g. on an obstacle, asphalt road that could have traffic, etc).
- **Inference Time:** The time between sending a query to the large VLM and it returning a reference path. This includes network latencies.

5.4.3.4 Navigation Testing Scenarios

We compare all the methods across four types of scenarios: 1. Indoor corridors, 2. Indoor scenarios with people, 3. Outdoor scenarios with multiple terrains, 4. Outdoor crosswalks. Additionally, we illustrate the addition of behaviors for a new scenario and showcase the resulting robot behavior.

5.4.3.5 Analysis and Comparisons

The results of our qualitative and quantitative comparisons are shown in Fig. 5.7, and Table 5.4, respectively. The results of our ablation studies are shown in Table 5.3.

Ablation Study: We analyze the effects of our multi-modal visual marking (MMVM), and context-based prompting on generating the reference path by removing these components individually from CoNVOI. Next, we compare the generated reference path with human-provided reference paths (see Fig. 5.8). First, removing MMVM, the marked image I_t^{Mark} would contain numbers on obstacles (walls, people, etc.). Therefore, while choosing the numbers for the reference path, the VLM must also ensure that they do not lie on obstacles (thus being responsible for avoiding them). However, we observe that this results in high reference path errors, and % of unacceptable paths as a subset of the chosen points often lie on obstacles in indoor scenarios

Scenario	Method	Reference Path Error ↓	Cosine Similarity ↑	% Unacceptable Paths ↓	Inference Time ↓
Scen. 1	Gemini w/o MMVM w. CbP	1.442	0.926	0.857	5.24
	GPT4 w/o MMVM w. CbP	1.576	0.913	0.913	4.86
	Gemini w. MMVM w/o CbP	0.315	0.995	0.0	4.78
	GPT4 w. MMVM w/o CbP	0.335	0.993	0.0	4.10
	Gemini w. CoNVOI	0.202	0.996	0.0	3.76
	GPT4 w. CoNVOI	0.187	0.995	0.0	3.48
Scen. 2	Gemini w/o MMVM w. CbP	1.425	0.914	0.786	5.11
	GPT4 w/o MMVM w. CbP	2.325	0.802	0.798	4.69
	Gemini w. MMVM w/o CbP	0.484	0.989	0.278	4.91
	GPT4 w. MMVM w/o CbP	0.589	0.988	0.223	4.46
	Gemini w. CoNVOI	0.444	0.981	0.111	3.55
	GPT4 w. CoNVOI	0.456	0.991	0.167	3.28
Scen. 3	Gemini w/o MMVM w. CbP	0.891	0.973	0.433	6.35
	GPT4 w/o MMVM w. CbP	0.865	0.977	0.366	5.60
	Gemini w. MMVM w/o CbP	0.817	0.978	0.291	6.28
	GPT4 w. MMVM w/o CbP	0.872	0.971	0.125	5.46
	Gemini w. CoNVOI	0.979	0.970	0.125	4.88
	GPT4 w. CoNVOI	0.785	0.975	0.083	4.15
Scen. 4	Gemini w/o MMVM w. CbP	1.210	0.957	0.285	6.36
	GPT4 w/o MMVM w. CbP	1.205	0.966	0.190	4.89
	Gemini w. MMVM w/o CbP	1.229	0.938	0.223	5.98
	GPT4 w. MMVM w/o CbP	1.159	0.962	0.166	4.95
	Gemini w. CoNVOI	1.205	0.977	0.055	5.26
	GPT4 w. CoNVOI	0.819	0.978	0.055	4.83

Table 5.3: Ablation studies comparing the impact of removing multi-modal visual marking (MMVM) and context-based prompting (CbP) from our formulation on generating reference paths. We also compare the use of two state-of-the-art large VLMs: GPT-4v [237] and Gemini [240]. Our results show that MMVM and CbP significantly improve the VLM’s performance in generating human-like reference paths.

Scenario	Method	Fréchet Dist. ↓	Norm. Traj. Length ≈ 1	Mean Velocity ↑
Scen. 1	DWA [29]	0.864	1.082	0.395
	Frozone [54]	0.577	1.239	0.392
	CoNVOI (ours)	0.521	1.207	0.289
Scen. 2	DWA [29]	1.087	1.105	0.390
	Frozone [54]	0.866	1.187	0.388
	CoNVOI (ours)	0.709	1.194	0.276
Scen. 3	DWA [29]	2.475	1.067	0.485
	PSP-Net [219]	2.188	1.121	0.458
	GA-Nav [18]	1.957	1.160	0.473
	CoNVOI (ours)	1.083	1.213	0.238
Scen. 4	DWA [29]	2.095	1.091	0.491
	PSP-Net [219]	2.068	1.089	0.455
	GA-Nav [18]	2.157	1.112	0.466
	CoNVOI (ours)	0.863	1.237	0.243

Table 5.4: Performance comparisons when a robot uses CoNVOI DWA motion planner [29] that performs only obstacle avoidance and goal-reaching, Frozone [54], a socially-compliant indoor navigation method, and PSP-Net [219] and GA-Nav [18], two semantic segmentation-based outdoor navigation methods. CoNVOI generates trajectories that are most similar to human teleoperated trajectories (lowest Fréchet distance). Qualitatively, CoNVOI performs similar to social and outdoor navigation methods in a zero-shot manner and is applicable to both indoor and outdoor environments.

(1 and 2). Additionally, the cosine similarity when compared with human GT is low, indicating that the paths are not generally in the direction preferred by a human. In outdoor environments, removing MMVM does not affect the metrics as adversely, since these scenarios are expansive and less constrained by humans. However, the % of unacceptable paths is higher as the returned reference paths some times intersect with walking pedestrian obstacles. The inference rate is also increased since the VLM must now pay attention to more regions before choosing points for the reference path. We observe that current VLMs cannot be used for zero-shot obstacle avoidance reliably.

In the second ablation study, we removed context-based prompting (CbP) and used a single

detailed prompt describing the rules for navigating all associated scenarios. We observe that this leads to a performance comparable to CoNVOI in terms of the path error and cosine similarity. However, there is a considerable increase in the VLM’s inference time. This could be due to the higher number of tokens the VLM must process given a detailed text prompt. Additionally, we observe a significant increase in the % of unacceptable paths in scenarios 2 and 4, where certain intricate behaviors (avoid moving in-between people, navigate on crosswalk) are expected. Such behaviors may be harder to interpret accurately when included with behaviors for other scenarios in the detailed prompt. We observe the lowest path errors, % of unacceptable paths, inference time, and highest cosine similarity for CoNVOI (with MMVM, CbP) in all cases. Since using CoNVOI with GPT-4v [237] has the least inference rate along with the best reference paths, we use it for comparison with other navigation methods in Table 5.4.

Comparison with Navigation Methods: We compare all the navigation methods relative to human teleoperation, which is considered the *best* trajectory the robot could take in its current context. We observe that CoNVOI’s zero-shot path resemble the human teleoperated path the closest (lowest Fréchet distance in Table 5.4) in all the scenarios (see Fig. 5.2, and 5.7).

We observe that CoNVOI typically leads to a higher normalized trajectory length due to higher deviations from the goal to follow the context-based navigation behavior, similar to the human teleoperated path. Additionally, CoNVOI has a low mean velocity due to its limiting maximum velocity ($v_{max} = 0.3$ m/s) set to ensure uninterrupted navigation, due to the high inference time current VLMs require. However, we highlight that CoNVOI demonstrates all these context-based behaviors in a zero-shot manner, utilizing simple text and visual prompts. Unlike existing model-based and deep learning-based methods, it has demonstrated wide applicability across various indoor and outdoor environments.

Handling Novel Scenarios: We also highlight that new types of scenarios can be easily added to CoNVOI without any reformulation. We add a scenario with a detour sign and instruct CoNVOI to deviate the robot in the direction of the detour sign’s arrow. We observe that CoNVOI immediately adapts to the new scenario, and navigates to its goal behind the sign (Fig. 5.2 [bottom]). This signifies how CoNVOI can use VLMs’ complex visual understanding and reasoning capabilities for navigation in the real-world.

Extrapolation Benefits: We observe that CoNVOI’s extrapolation helps avoid unnecessarily requerying the large VLM in environments where the context does not change until the goal (e.g. in corridors, straight pavements). On average, extrapolation helps reduce the requerying by $\sim 50\%$ in such environments. In dynamic scenarios with walking humans, in the presence of several terrains (extrapolated point could lie in unpaved terrain), the VLM is requered to guide the robot.

VLM Hallucinations: Since during navigation the large VLM is typically fed with a sequence of *similar*-looking images and text prompts, we observe that it often starts to hallucinate and return reference path numbers in a certain pattern (e.g. [1, 2, 3, 4, 5], or [1, 2, 1, 2]) without considering the scenario. This occurs when the VLM uses its recent memory to perform the current task. To alleviate hallucinations, we add a phrase to the text prompt to instruct the VLM that it must consider the current query as a new task and disregard past inputs and outputs.

Wavy Trajectories: We observe that when there are several correct paths that satisfy the behavior specified in the context-based text prompt, the VLM may return any one of them. This could sometimes result in a wavy trajectory (as shown in Fig. 5.2 on the crosswalk, while passing the barricade). This is primarily due to the spacing parameter d_{col} between the numbers marked in the image, which affects the spatial resolution of the reference paths. However, decreasing d_{col}

leads to more numbers annotated on the image, which leads to higher inference rates to process, similar to removing MMVM in Table 5.3. Such limitations could be alleviated with faster large VLMs in the future.

5.4.4 Limitations

Since CoNVOI uses a large VLM that is hosted remotely, its processing time and network latency affects the robot’s navigation and its maximum velocity. In outdoor environments, since network speeds also depend on weather conditions, we observe some rare instances where the robot does not receive a reference path, and uses the baseline planner to navigate. Additionally, this limits VLMs’ use in highly dynamic environments with crowds. However, such limitations can be overcome as large VLMs become faster, and locally executable. We also observe that outputs could be sensitive to vague terms in the text prompt (e.g. “*cross the street like a human*” instead of specifying “*stay on the crosswalk*”). Our current formulation does not use any global information like a top-down view of the environment, which would be needed to assess factors like where to cross the road, when the crosswalk is not visible from the robot’s camera. Finally, CoNVOI uses the manually framed text prompts. It would be beneficial to allow the VLM to automatically frame such prompts in the future.

5.5 Summary

In this chapter, we presented two methods that operate in both indoor and outdoor environments. The first method, MIM dealt with challenges in sensing such as detecting transparent obstacles, thin pliable curtains, grass, etc. MIM introduced a novel obstacle representation designed to

enhance autonomous robot navigation. Based on the intensity of reflected points from point clouds, MIM effectively characterized obstacles by their height, solidity, and opacity. Also, we presented an adaptive inflation technique that further refines navigation cost maps by considering obstacle solidity and available free space. We demonstrated significant improvements in navigation metrics such as success rates, trajectory lengths, and F-scores, validating our proposed approach.

The second method addressed the problem of adapting the robot’s navigation behaviors based on its environmental context. We presented CoNVOI, a novel method to use compact and large VLMs for indoor and outdoor navigation without any additional training or fine-tuning. CoNVOI utilized a novel multi-modal visual marking scheme, and a context-based prompting method to achieve complex behaviors (e.g. avoiding moving in-between groups of humans, interpreting signs, etc) in various indoor and outdoor scenarios. CoNVOI also used an extrapolation method to avoid unnecessary requerying of the large VLM that reduces the queries by up to 50% in certain scenarios. Both methods have been implemented and demonstrated on a real Boston Dynamics Spot and Turtlebot 2.

Using MIM and CoNVOI enables mobile robots to operate in both indoor and outdoor environments. However, their applicability has not been evaluated in highly dynamic environments. MIMs can be integrated with the crowd navigation methods in chapters 2, and 3. However, CoNVOI cannot be immediately deployed in such scenarios due to the high inference times of present-day state-of-the-art VLMs. As VLMs continue to become faster and more capable, CoNVOI’s general context-based prompting and visual marking can enable them to be quickly deployed for robot navigation tasks.

Chapter 6: Conclusion

This dissertation significantly advanced the autonomous navigation capabilities of mobile robots in various indoor and outdoor environments. Challenges such as navigating constrained indoor environments with crowds of varying densities, unstructured off-road terrains with different surface properties (e.g. deformable sand, rocks, fallen tree branches, etc), and dense vegetation. Finally, we discuss general methods that solve sensing challenges (e.g. detecting transparent obstacles, tall, thin, pliable obstacles, etc), and address context-aware navigation behaviors in indoors and outdoors.

Chapters 2 and 3 discussed methods for crowd navigation. The presented formulations evolved from considering dynamic humans in social scenarios only as individual obstacles to detecting groups in a crowd and treating them as one cohesive unit to analyze their characteristics. They also evolved from being purely learning-based to more practical model-based and learning-based hybrids. Our methods also progressively became less constrained in choosing collision-free velocities. For instance, the PFZs in Frozone (section 2.4) were formulated as *group* PFZs in CoMet 3.4, which made more free space available to the robot to traverse. We demonstrated the practical feasibility of all our methods using real robots in real-world crowd scenarios.

A few limitations related to Deep Reinforcement Learning (section 2.3) remain. For instance, the sim-to-real gap and poor explainability of these models are still prevalent, although our work

attempts to reduce the sim-to-real gap using realistic 3D simulations. An important, parallel research direction to crowd navigation is Human Robot Interaction (HRI). Extensive research on how humans react when walking alongside mobile robots, and how their responses differ when compared to avoiding other humans are required. Further, methods that lead to improved navigation behaviors that account for the surrounding context *with* dynamic humans (beyond our static experiments in CoNVOI in chapter 5) are required. These methods must also be formulated to have low computational overhead, and operate in real-time on edge computing devices.

Chapter 4 discussed TerraPN and VERN for unstructured outdoor navigation: TerraPN for wheeled, and VERN for legged robots. An important feature of both methods was that they were based on self-supervised and few-shot learning, respectively, and hence there was no need for human annotated datasets for training their perception components. Further, these methods required far less training data compared to traditional supervised learning-based methods to exhibit the same navigation behaviors. Both methods operate in real-time on laptop-grade or edge computing GPUs onboard robots in outdoor environments.

An important avenue for future research in outdoor navigation is perception that is robust to changes in the environment such as extreme lighting and weather conditions. Moreover, there is a need to develop learning methods that can generalize the training from one environment to another without high model complexity and computational resources. For example, the robot’s perception model must infer that an oak tree during noon, and a short maple tree at dusk should both be viewed as the same from a navigation standpoint. One possible direction to achieve this is by fusing data from different sensor modalities, each of which help detect a specific features of the environment robustly. Another interesting direction, as recent research on large models have demonstrated is achieving generalization by using large models trained with hundreds of billions

of parameters and data points.

Chapter 5 explored the use of such large models for real-time navigation, along with a generalized method based on point cloud intensities to sense complex obstacles in both indoor and outdoor environments. The results the proposed methods are promising. However, significant strides in multi-modal large models are required to make navigation truly environment-agnostic. These models must be improved to be hosted and executed locally for faster navigation, and handling dynamic obstacles. Once realized, these models will enable mobile robots to exhibit truly adaptive, intelligent, and human-like behaviors.

We believe that through the research conducted in this dissertation, we have overcome many limitations and identified new theoretical, and practical ones. The presented research also spans the distinct challenges posed by real-world indoor and outdoor settings. All the presented methods are designed for real-world applicability and realizability, and the practical challenges in their implementation have also been reported, which is crucial for advancing robotics research in autonomous navigation across various environments.

Bibliography

- [1] Ahmed Rhiat, Lamine Chalal, and Allal Saadane. A smart warehouse using robots and drone to optimize inventory management. In *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 1*, pages 475–483. Springer, 2022.
- [2] Anna MH Abrams, Pia SC Dautzenberg, Carla Jakobowsky, Stefan Ladwig, and Astrid M Rosenthal-Von Der Pütten. A theoretical and empirical reflection on technology acceptance models for autonomous delivery robots. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 272–280, 2021.
- [3] Jane Holland, Liz Kingston, Conor McCarthy, Eddie Armstrong, Peter O’Dwyer, Fionn Merz, and Mark McConnell. Service robots in the healthcare sector. *Robotics*, 10(1):47, 2021.
- [4] Juan Angel Gonzalez-Aguirre, Ricardo Osorio-Oliveros, Karen L Rodríguez-Hernández, Javier Lizárraga-Iturralde, Ruben Morales Menendez, Ricardo A Ramírez-Mendoza, Mauricio Adolfo Ramírez-Moreno, and Jorge de Jesus Lozoya-Santos. Service robots: Trends and technology. *Applied Sciences*, 11(22):10702, 2021.
- [5] Josip Balen, Davor Damjanovic, Petar Maric, Kresimir Vdovjak, Matej Arlovic, and Goran Martinovic. Firebot-an autonomous surveillance robot for fire prevention, early detection and extinguishing. In *2023 15th International Conference on Computer and Automation Engineering (ICCAE)*, pages 400–405. IEEE, 2023.
- [6] Chao Cheng, Jun Fu, Hang Su, and Luquan Ren. Recent advancements in agriculture robots: Benefits and challenges. *Machines*, 11(1):48, 2023.
- [7] Philip Arm, Gabriel Waibel, Jan Preisig, Turcan Tuna, Ruyi Zhou, Valentin Bickel, Gabriela Ligeza, Takahiro Miki, Florian Kehl, Hendrik Kolvenbach, et al. Scientific exploration of challenging planetary analog environments with a team of legged robots. *Science robotics*, 8(80):eade9548, 2023.
- [8] Gongcheng Wang, Weidong Wang, Pengchao Ding, Yueming Liu, Han Wang, Zhenquan Fan, Hua Bai, Zhu Hongbiao, and Zhijiang Du. Development of a search and rescue robot

- system for the underground building environment. *Journal of Field Robotics*, 40(3):655–683, 2023.
- [9] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything. *arXiv e-prints*, page arXiv:2304.02643, April 2023.
 - [10] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.
 - [11] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, yxNONG, Adam Hogan, lorenzomamma, AlexWang1900, Ayush Chaurasia, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Durgesh, Francisco Ingham, Frederik, Guilhen, Adrien Colmagro, Hu Ye, Jacobsolawetz, Jake Poznanski, Jiacong Fang, Junghoon Kim, Khiem Doan, and Lijun Yu. ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration, January 2021.
 - [12] N. Wojke, A. Bewley, and D. Paulus. Simple Online and Realtime Tracking with a Deep Association Metric. *ArXiv e-prints*, March 2017.
 - [13] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *arXiv e-prints*, page arXiv:1812.08008, December 2018.
 - [14] Javier Orozco, Shaogang Gong, and Tao Xiang. Head pose classification in crowded scenes. 01 2009.
 - [15] T. Mittal, P. Guhan, U. Bhattacharya, R. Chandra, A. Bera, and D. Manocha. Emoticon: Context-aware multimodal emotion recognition using frege’s principle. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14222–14231, 2020.
 - [16] Haiyang Mei, Xin Yang, Yang Wang, Yuanyuan Liu, Shengfeng He, Qiang Zhang, Xiaopeng Wei, and Rynson W.H. Lau. Don’t hit me! glass detection in real-world scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
 - [17] Haiyang Mei, Xin Yang, Yang Wang, Yuanyuan Liu, Shengfeng He, Qiang Zhang, Xiaopeng Wei, and Rynson WH Lau. Don’t hit me! glass detection in real-world scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3687–3696, 2020.
 - [18] Tianrui Guan, Divya Kothandaraman, Rohan Chandra, Kasun Weerakoon Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Ganav: Efficient terrain segmentation for robot navigation in unstructured outdoor environments, 2021.

- [19] Tien Do, Ondrej Miksik, Joseph DeGol, Hyun Soo Park, and Sudipta N. Sinha. Learning to detect scene landmarks for camera localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11132–11142, June 2022.
- [20] Xiwu Zhang, Lei Wang, and Yan Su. Visual place recognition: A survey from deep learning perspective. *Pattern Recognition*, 113:107760, 2021.
- [21] S Burak Gokturk, Hakan Yalcin, and Cyrus Bamji. A time-of-flight depth sensor-system description, issues and solutions. In *2004 conference on computer vision and pattern recognition workshop*, pages 35–35. IEEE, 2004.
- [22] Rostam Affendi Hamzah, Haidi Ibrahim, et al. Literature survey on stereo vision disparity map algorithms. *Journal of Sensors*, 2016, 2016.
- [23] Xianyu Qi, Wei Wang, Mei Yuan, Yuliang Wang, Mingbo Li, Lin Xue, and Yingpin Sun. Building semantic grid maps for domestic robot navigation. *International Journal of Advanced Robotic Systems*, 17(1):1729881419900066, 2020.
- [24] Hrudaya Kumar Tripathy, Sushruta Mishra, Hiren Kumar Thakkar, and Deepak Rai. Care: A collision-aware mobile robot navigation in grid environment using improved breadth first search. *Computers & Electrical Engineering*, 94:107327, 2021.
- [25] Daniele De Gregorio and Luigi Di Stefano. Skimap: An efficient mapping framework for robot navigation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2569–2576. IEEE, 2017.
- [26] Francesco Verdoja, Jens Lundell, and Ville Kyrki. Deep network uncertainty maps for indoor navigation. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 112–119. IEEE, 2019.
- [27] Kasun Weerakoon, Adarsh Jagan Sathyamoorthy, Utsav Patel, and Dinesh Manocha. Terp: Reliable planning in uneven outdoor environments using deep reinforcement learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9447–9453. IEEE, 2022.
- [28] Henry Roncancio, Marcelo Becker, Alberto Broggi, and Stefano Cattani. Traversability analysis using terrain mapping and online-trained terrain type classifier. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1239–1244, 2014.
- [29] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, March 1997.
- [30] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *2015 European Control Conference (ECC)*, pages 3352–3357, 2015.

- [31] J. van den Berg, Ming Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, May 2008.
- [32] Y.K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.
- [33] Jur P. van den Berg, Stephen J. Guy, M. Chiao Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *ISRR*, 2009.
- [34] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1709.10082, Sep 2017.
- [35] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Fully Distributed Multi-Robot Collision Avoidance via Deep Reinforcement Learning for Safe and Efficient Navigation in Complex Scenarios. *arXiv e-prints*, page arXiv:1808.03841, Aug 2018.
- [36] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *ICRA*, pages 285–292. IEEE, 2017.
- [37] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan How. Socially aware motion planning with deep reinforcement learning. pages 1343–1350, 09 2017.
- [38] Tingxiang Fan, Xinjing Cheng, Jia Pan, Dinesh Manocha, and Ruigang Yang. CrowdMove: Autonomous Mapless Navigation in Crowded Scenarios. *arXiv e-prints*, page arXiv:1807.07870, Jul 2018.
- [39] Sriram Siva, Maggie Wigness, John Rogers, and Hao Zhang. Robot adaptation to unstructured terrains by joint representation and apprenticeship learning. 7 2019.
- [40] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.
- [41] S. Shimoda, Y. Kuroda, and K. Iagnemma. Potential field navigation of high speed unmanned ground vehicles on uneven terrain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2828–2833, 2005.
- [42] L. Tai, J. Zhang, M. Liu, and W. Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *ICRA*, pages 1111–1117, May 2018.
- [43] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *ICRA*, pages 6015–6022. IEEE, 2019.
- [44] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks. *ArXiv e-prints*, March 2018.

- [45] Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*, 2018.
- [46] Kacper Piotr Kielak. Do recent advancements in model-based deep reinforcement learning really improve data efficiency? 2019.
- [47] Wenshuai Zhao, Jorge Pena Queralta, Li Qingqing, and Tomi Westerlund. Towards closing the sim-to-real gap in collaborative multi-robot deep reinforcement learning. In *2020 5th International conference on robotics and automation engineering (ICRAE)*, pages 7–12. IEEE, 2020.
- [48] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.
- [49] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access*, 9:153171–153187, 2021.
- [50] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at <http://planning.cs.uiuc.edu/>.
- [51] Salim Belkhou, Adel Azzouz, Maarouf Saad, Chahe Nerguizian, and Vahe Nerguizian. A novel approach for mobile robot navigation with dynamic obstacles avoidance. *Journal of Intelligent and Robotic Systems*, 44:187–201, 2005.
- [52] Anish Pandey, Shalini Pandey, and DR Parhi. Mobile robot navigation and obstacle avoidance techniques: A review. *Int Rob Auto J*, 2(3):00022, 2017.
- [53] Adarsh Jagan Sathyamoorthy, Jing Liang, Utsav Patel, Tianrui Guan, Rohan Chandra, and Dinesh Manocha. DenseCAvoid: Real-time Navigation in Dense Crowds using Anticipatory Behaviors. *arXiv e-prints*, page arXiv:2002.03038, February 2020.
- [54] Adarsh Jagan Sathyamoorthy, Utsav Patel, Tianrui Guan, and Dinesh Manocha. Frozone: Freezing-free, pedestrian-friendly navigation in human crowds. *IEEE Robotics and Automation Letters*, 5(3):4352–4359, 2020.
- [55] Adarsh Jagan Sathyamoorthy, Utsav Patel, Moumita Paul, Nithish K Sanjeev Kumar, Yash Savle, and Dinesh Manocha. Comet: Modeling group cohesion for socially compliant robot navigation in crowded scenes. *IEEE Robotics and Automation Letters*, 7(2):1008–1015, 2022.
- [56] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul A. Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *DARS*, 2010.

- [57] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers. A hybrid collision avoidance method for mobile robots. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 2, pages 1238–1243 vol.2, May 1998.
- [58] Y. Kim, J. Jang, and S. Yun. End-to-end deep learning for autonomous navigation of mobile robot. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, Jan 2018.
- [59] 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 *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1527–1533, May 2017.
- [60] A. Bera and D. Manocha. Reach - realtime crowd tracking using a hybrid motion model. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 740–747, May 2015.
- [61] Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *IROS*, pages 3052–3059. IEEE, 2018.
- [62] Sahil Narang, Andrew Best, Sean Curtis, and Dinesh Manocha. Generating pedestrian trajectories consistent with the fundamental diagram based on physiological and psychological factors. *PLOS ONE*, 10:e0117856, 04 2015.
- [63] A. Bera, T. Randhavane, R. Prinja, and D. Manocha. Sociosense: Robot navigation amongst pedestrians with social and psychological constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7018–7025, Sep. 2017.
- [64] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan How. Socially aware motion planning with deep reinforcement learning. pages 1343–1350, 09 2017.
- [65] P. Trautman and A. Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803, Oct 2010.
- [66] Pete Trautman, Jeremy Ma, Richard M. Murray, and Andreas Krause. Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation. *The International Journal of Robotics Research*, 34(3):335–356, 2015.
- [67] Andrey Rudenko, Luigi Palmieri, and Kai Oliver Arras. Predictive planning for a mobile robot in human environments. In *Proceedings of the Workshop on AI Planning and Robotics: Challenges and Methods (at ICRA 2017)*, Singapore, 2017.
- [68] Tingxiang Fan, Xinjing Cheng, Jia Pan, Pinxin Long, Wenxi Liu, Ruigang Yang, and Dinesh Manocha. Getting robots unfrozen and unlost in dense pedestrian crowds. *IEEE Robotics and Automation Letters*, PP:1–1, 01 2019.

- [69] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *ArXiv e-prints*, November 2013.
- [70] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [71] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *ArXiv e-prints*, June 2015.
- [72] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. *ArXiv e-prints*, March 2017.
- [73] Anton Milan, S Hamid Rezatofighi, Anthony Dick, Ian Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [74] Seung-Hwan Bae and Kuk-Jin Yoon. Confidence-based data association and discriminative deep appearance learning for robust online multi-object tracking. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):595–610, 2018.
- [75] Qi Chu, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, Bin Liu, and Nenghai Yu. Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism. 2017.
- [76] Kuan Fang, Yu Xiang, and Silvio Savarese. Recurrent autoregressive networks for online multi-object tracking. *arXiv preprint arXiv:1711.02741*, 2017.
- [77] Samuel Murray. Real-time multiple object tracking-a study on the importance of speed. *arXiv preprint arXiv:1709.03572*, 2017.
- [78] Anton Milan, Stefan Roth, and Konrad Schindler. Continuous energy minimization for multitarget tracking. *IEEE transactions on pattern analysis and machine intelligence*, 36(1):58–72, 2013.
- [79] Chanh Kim, Fuxin Li, Arridhana Ciptadi, and James M Rehg. Multiple hypothesis tracking revisited. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4696–4704, 2015.
- [80] Roberto Henschel, Laura Leal-Taixe, Daniel Cremers, and Bodo Rosenhahn. Fusion of head and full-body detectors for multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1428–1437, 2018.
- [81] Hao Sheng, Li Hao, Jiahui Chen, Yang Zhang, and Wei Ke. Robust local effective matching model for multi-target tracking. In *Pacific Rim Conference on Multimedia*, pages 233–243. Springer, 2017.
- [82] Aniket Bera and Dinesh Manocha. Realtime multilevel crowd tracking using reciprocal velocity obstacles. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 4164–4169. IEEE, 2014.

- [83] Aniket Bera, Nico Galoppo, Dillon Sharlet, Adam Lake, and Dinesh Manocha. Adapt: real-time adaptive pedestrian tracking for crowded scenes. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1801–1808. IEEE, 2014.
- [84] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool. You’ll never walk alone: Modeling social behavior for multi-target tracking. In *2009 IEEE 12th International Conference on Computer Vision*, pages 261–268, Sept 2009.
- [85] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1345–1352. IEEE, 2011.
- [86] Stéphanie Lefèvre, Christian Laugier, and Javier Ibañez-Guzmán. Exploiting map information for driver intention estimation at road intersections. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 583–588. IEEE, 2011.
- [87] Simon Danielsson, Lars Petersson, and Andreas Eidehall. Monte carlo based threat assessment: Analysis and improvements. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 233–238. IEEE, 2007.
- [88] Jonas Firl, Hagen Stübing, Sorin A Huss, and Christoph Stiller. Predictive maneuver evaluation for enhancement of car-to-x mobility data. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 558–564. IEEE, 2012.
- [89] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering*, 82(Series D):35–45, 1960.
- [90] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [91] Fang-Chieh Chou, Tsung-Han Lin, Henggang Cui, Vladan Radosavljevic, Thi Nguyen, Tzu-Kuo Huang, Matthew Niedoba, Jeff Schneider, and Nemanja Djuric. Predicting motion of vulnerable road users using high-definition maps and efficient convnets. 2018.
- [92] Hengchin Yeh, Sean Curtis, Sachin Patil, Jur van den Berg, Dinesh Manocha, and Ming Lin. Composite agents. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 39–47, 2008.
- [93] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.
- [94] Stephen J Guy, Jur Van Den Berg, Wenxi Liu, Rynson Lau, Ming C Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Transactions on Graphics (TOG)*, 31(6):1–11, 2012.
- [95] Sujeong Kim, Stephen J. Guy, Wenxi Liu, David Wilkie, Rynson W.H. Lau, Ming C. Lin, and Dinesh Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *The International Journal of Robotics Research*, 34(2):201–217, 2015.

- [96] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee. Intention-aware online pomdp planning for autonomous driving in a crowd. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 454–460, May 2015.
- [97] Y. Luo, P. Cai, A. Bera, D. Hsu, W. S. Lee, and D. Manocha. Porca: Modeling and planning for autonomous driving among many pedestrians. *IEEE Robotics and Automation Letters*, 3(4):3418–3425, Oct 2018.
- [98] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1706.09829, Jun 2017.
- [99] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots. *arXiv e-prints*, page arXiv:1609.07910, Sep 2016.
- [100] Tingxiang Fan, Xinjing Cheng, Jia Pan, Pinxin Long, Wenxi Liu, Ruigang Yang, and Dinesh Manocha. Getting robots unfrozen and unlost in dense pedestrian crowds. *IEEE Robotics and Automation Letters*, 4(2):1178–1185, 2019.
- [101] B. Lötjens, M. Everett, and J. P. How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668, May 2019.
- [102] P. Trautman, J. Ma, R. M. Murray, and A. Krause. Robot navigation in dense human crowds: the case for cooperation. In *2013 IEEE International Conference on Robotics and Automation*, pages 2153–2160, May 2013.
- [103] C. Yu and C. Wang. Collision- and freezing-free navigation in dynamic environments using learning to search. In *2012 Conference on Technologies and Applications of Artificial Intelligence*, pages 151–156, Nov 2012.
- [104] Andrew Best, Sahil Narang, Sean Curtis, and Dinesh Manocha. Densesense: Interactive crowd simulation using density-dependent filters. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 97–102. Eurographics Association, 2014.
- [105] Aniket Bera, Tanmay Randhavane, Austin Wang, Dinesh Manocha, Emily Kubin, and Kurt Gray. Classifying group emotions for socially-aware autonomous vehicle navigation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.
- [106] Rohan Chandra, Uttaran Bhattacharya, Christian Roncal, Aniket Bera, and Dinesh Manocha. Robusttp: End-to-end trajectory prediction for heterogeneous road-agents in dense traffic with noisy sensor inputs. *arXiv preprint arXiv:1907.08752*, 2019.
- [107] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv e-prints*, page arXiv:1707.06347, Jul 2017.

- [108] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [109] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [110] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS’99*, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.
- [111] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.
- [112] Rohan Chandra, Uttaran Bhattacharya, Aniket Bera, and Dinesh Manocha. Densepeds: Pedestrian tracking in dense crowds using front-rvo and sparse features. *arXiv preprint arXiv:1906.10313*, 2019.
- [113] Liang Mao. Agent-based simulation for weekend-extension strategies to mitigate influenza outbreaks. *BMC Public Health*, 11(1):522, Dec 2011.
- [114] S. Kumar, J. J. Grefenstette, D. Galloway, S. M. Albert, and D. S. Burke. Policies to reduce influenza in the workplace: impact assessments using an agent-based model. *Am J Public Health*, 103(8):1406–1411, Aug 2013.
- [115] T. Timpka, H. Eriksson, E. Holm, M. Stromgren, J. Ekberg, A. Spreco, and Dahlstrom. Relevance of workplace social mixing during influenza pandemics: an experimental modelling study of workplace cultures. *Epidemiol. Infect.*, 144(10):2031–2042, July 2016.
- [116] G. J. Milne, J. K. Kelso, H. A. Kelly, S. T. Huband, and J. McVernon. A small community model for the transmission of infectious diseases: comparison of school closure as an intervention in individual-based models of an influenza pandemic. *PLoS ONE*, 3(12):1–7, 2008.
- [117] S. Cauchemez, A. J. Valleron, P. Y. Boëlle, A. Flahault, and N. M. Ferguson. Estimating the impact of school closure on influenza transmission from Sentinel data. *Nature*, 452(7188):750–754, Apr 2008.
- [118] Cong T. Nguyen, Yuris Mulya Saputra, Nguyen Van Huynh, Ngoc-Tan Nguyen, Tran Viet Khoa, Bui Minh Tuan, Diep N. Nguyen, Dinh Thai Hoang, Thang X. Vu, Eryk Dutkiewicz, Symeon Chatzinotas, and Bjorn Ottersten. Enabling and Emerging Technologies for Social Distancing: A Comprehensive Survey. *arXiv e-prints*, page arXiv:2005.02816, May 2020.
- [119] Liang He, Jia Pan, Sahil Narang, and Dinesh Manocha. Dynamic group behaviors for interactive crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 139–147. Eurographics Association, 2016.

- [120] Stefania Bandini, Andrea Gorrini, Lorenza Manenti, and Giuseppe Vizzari. Crowd and pedestrian dynamics: Empirical investigation and simulation. In *Proceedings of Measuring Behavior*, 08 2012.
- [121] N. Ashton, M. E. Shaw, and Annette Pearce Worsham. Affective reactions to interpersonal distances by friends and strangers. *Bulletin of the psychonomic society*, 15:306–308, 1980.
- [122] S. Karau and K. Williams. The effects of group cohesiveness on social loafing and social compensation. *Group Dynamics: Theory, Research, and Practice*, 1:156–168, 1997.
- [123] ES Knowles. Boundaries around group interaction: the effect of group size and member status on boundary permeability. *Journal of personality and social psychology*, 26(3):327—331, June 1973.
- [124] Edward T. Hall, Ray L. Birdwhistell, Bernhard Bock, Paul Bohannon, A. Richard Diebold, Marshall Durbin, Munro S. Edmonson, J. L. Fischer, Dell Hymes, Solon T. Kimball, Weston La Barre, , J. E. McClellan, Donald S. Marshall, G. B. Milner, Harvey B. Sarles, George L. Trager, and Andrew P. Vayda. Proxemics [and comments and replies]. *Current Anthropology*, 9(2/3):83–108, 1968.
- [125] Janelle Wagnild and Cara M. Wall-Scheffler. Energetic consequences of human sociality: Walking speed choices among friendly dyads. *PLOS ONE*, 8(10):1–6, 10 2013.
- [126] J. Shao, C. C. Loy, and X. Wang. Scene-independent group profiling in crowd. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2227–2234, 2014.
- [127] Dongfang Yang, Ekim Yurtsever, Vishnu Renganathan, Keith Redmill, and Umit Ozguner. A vision-based social distancing and critical density detection system for covid-19, 07 2020.
- [128] Narinder Singh Pun, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques. *arXiv e-prints*, page arXiv:2005.01385, May 2020.
- [129] Isha Ghodgaonkar, Subhankar Chakraborty, Vishnu Banna, Shane Allcroft, Mohammed Metwally, Fischer Bordwell, Kohsuke Kimura, Xinxin Zhao, Abhinav Goel, Caleb Tung, Akhil Chinnakotla, Minghao Xue, Yung-Hsiang Lu, Mark Daniel Ward, Wei Zakharov, David S. Ebert, David M. Barbarash, and George K. Thiruvathukal. Analyzing Worldwide Social Distancing through Large-Scale Computer Vision. *arXiv e-prints*, page arXiv:2008.12363, August 2020.
- [130] Robin R. Murphy, Vignesh Babu Manjunath Gandudi, and Justin Adams. Applications of Robots for COVID-19 Response. *arXiv e-prints*, page arXiv:2008.06976, August 2020.
- [131] Tingxiang Fan, Zhiming Chen, Xuan Zhao, Jing Liang, Cong Shen, Dinesh Manocha, Jia Pan, and Wei Zhang. Autonomous Social Distancing in Urban Environments using a Quadruped Robot. *arXiv e-prints*, page arXiv:2008.08889, August 2020.

- [132] Rohan Chandra, Uttaran Bhattacharya, Aniket Bera, and Dinesh Manocha. Densepeds: Pedestrian tracking in dense crowds using front-rvo and sparse features. *arXiv preprint arXiv:1906.10313*, 2019.
- [133] Wongun Choi, Yu-Wei Chao, Caroline Pantofaru, and Silvio Savarese. Discovering groups of people in images. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 417–433, Cham, 2014. Springer International Publishing.
- [134] A. K. Chandran, L. A. Poh, and P. Vadakkepat. Identifying social groups in pedestrian crowd videos. In *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*, pages 1–6, 2015.
- [135] R. M. Favaretto, L. L. Dihl, and S. R. Musse. Detecting crowd features in video sequences. In *2016 29th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 201–208, 2016.
- [136] Zeinab Ebrahimipour, Wanggen Wan, Ofelia Cervantes, Tianhang Luo, and Hidayat Ullah. Comparison of main approaches for extracting behavior features from crowd flow analysis. *ISPRS International Journal of Geo-Information*, 8(10), 2019.
- [137] M. Yang, L. Rashidi, A. S. Rao, S. Rajasegarar, M. Ganji, M. Palaniswami, and C. Leckie. Cluster-based crowd movement behavior detection. In *2018 Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, 2018.
- [138] Marco Cristani, Loris Bazzani, Giulia Paggetti, Andrea Fossati, Diego Tosato, Alessio Del Bue, Gloria Menegaz, and Vittorio Murino. Social interaction discovery by statistical analysis of f-formations. In *Proceedings of the British Machine Vision Conference*, pages 23.1–23.12. BMVA Press, 2011. <http://dx.doi.org/10.5244/C.25.23>.
- [139] Joseph P. Robinson, Ming Shao, Yue Wu, and Yun Fu. Families in the Wild (FIW): Large-Scale Kinship Image Database and Benchmarks. *arXiv e-prints*, page arXiv:1604.02182, April 2016.
- [140] G. Somanath and C. Kambhamettu. Can faces verify blood-relations? In *2012 IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pages 105–112, 2012.
- [141] S. M. Mathews, C. Kambhamettu, and K. E. Barner. “am i your sibling?” inferring kinship cues from facial image pairs. In *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5, 2015.
- [142] Mohammed Almuashi, S. Z. M. Hashim, D. Mohamad, M. H. Alkawaz, and A. Ali. Automated kinship verification and identification through human facial images: a survey. *Multimedia Tools and Applications*, 76:265–307, 2015.
- [143] Adarsh Jagan Sathyamoorthy, Utsav Patel, Yash Ajay Savle, Moumita Paul, and Dinesh Manocha. COVID-Robot: Monitoring Social Distancing Constraints in Crowded Scenarios. *arXiv e-prints*, page arXiv:2008.06585, August 2020.

- [144] Sean Curtis, Andrew Best, and Dinesh Manocha. Menge: A modular framework for simulating crowd movement. *Collective Dynamics*, 1:1–40, 2016.
- [145] Naga K Govindaraju, Ming C Lin, and Dinesh Manocha. Quick-cullide: Fast inter-and intra-object collision culling using graphics hardware. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, pages 59–66. IEEE, 2005.
- [146] Erving Goffman. *Encounters: Two studies in the sociology of interaction*. Ravenio Books, 1961.
- [147] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv e-prints*, page arXiv:1506.02640, June 2015.
- [148] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.
- [149] T. Baltrusaitis, A. Zadeh, Y. C. Lim, and L. Morency. Openface 2.0: Facial behavior analysis toolkit. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 59–66, 2018.
- [150] Károly Harsányi, Attila Kiss, András Majdik, and Tamas Sziranyi. A hybrid cnn approach for single image depth estimation: A case study. In Kazimierz Choroś, Marek Kopel, Elżbieta Kukla, and Andrzej Siemiński, editors, *Multimedia and Network Information Systems*, pages 372–381, Cham, 2019. Springer International Publishing.
- [151] Jing Liang, Utsav Patel, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Crowd-steer: Realtime smooth and collision-free robot navigation in densely crowded scenarios trained using high-fidelity simulation. In *IJCAI*, pages 4221–4228, 7 2020. Main track.
- [152] Anoop Aroor, Susan L Epstein, and Raj Korpan. Mengeros: A crowd simulation tool for autonomous robot navigation. In *AAAI 2017 Fall Symposium on Artificial Intelligence for Human-Robot Interaction*, 2017.
- [153] Kasun Weerakoon, Adarsh Jagan Sathyamoorthy, Utsav Patel, and Dinesh Manocha. Terp: Reliable planning in uneven outdoor environments using deep reinforcement learning, 2021.
- [154] Sriram Siva, Maggie Wigness, John Rogers, and Hao Zhang. Robot adaptation to unstructured terrains by joint representation and apprenticeship learning. 7 2019.
- [155] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system, 2020.
- [156] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319, 2021.

- [157] Zhu Bo, Gong Wei, Shi Shuo, and Song Shalei. A multi-wavelength canopy lidar for vegetation monitoring: System implementation and laboratory-based tests. *Procedia Environmental Sciences*, 10:2775–2782, 2011. 2011 3rd International Conference on Environmental Science and Information Application Technology ESIAT 2011.
- [158] Lorenz Wellhausen, Alexey Dosovitskiy, René Ranftl, Krzysztof Walas, Cesar Cadena, and Marco Hutter. Where should i walk? predicting terrain properties from images via self-supervised learning. *IEEE Robotics and Automation Letters*, 4(2):1509–1516, 2019.
- [159] Christopher A. Brooks and Karl D. Iagnemma. Self-supervised classification for planetary rover terrain sensing. In *2007 IEEE Aerospace Conference*, pages 1–9, 2007.
- [160] D. Kim, D. Carballo, J. Di Carlo, B. Katz, G. Bledt, B. Lim, and S. Kim. Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2464–2470, 2020.
- [161] Camilo Ordonez, Ryan Alicea, Brandon Rothrock, Kyle Ladyko, Jeremy Nash, Rohan Thakker, Shreyansh Daftry, Mario Harper, Emmanuel Collins, and Larry Matthies. Characterization and traversal of pliable vegetation for robot navigation. 11 2018.
- [162] Tianrui Guan, Zhenpeng He, Dinesh Manocha, and Liangjun Zhang. TTM: Terrain Traversability Mapping for Autonomous Excavators. *arXiv e-prints*, page arXiv:2109.06250, September 2021.
- [163] Fabian Schilling, Xi Chen, John Folkesson, and Patric Jensfelt. Geometric and visual terrain classification for autonomous mobile navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2684, 2017.
- [164] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Proceedings of 11th International Conference on Field and Service Robotics (FSR '17)*, pages 335 – 350, September 2017.
- [165] Adam Plevoy, Craig Knuth, Katie M. Popek, and Kapil D. Katyal. Complex Terrain Navigation via Model Error Prediction. *arXiv e-prints*, page arXiv:2111.09768, November 2021.
- [166] Dongshin Kim, Jie Sun, Sang Min Oh, J.M. Rehg, and A.F. Bobick. Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 518–525, 2006.
- [167] A. Lookingbill, J. Rogers, D. Lieb, J. Curry, and S. Thrun. Reverse optical flow for self-supervised adaptive autonomous robot navigation. *International Journal of Computer Vision*, 74:287–302, 01 2007.

- [168] Raia Hadsell, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144, 2009. Copyright: Copyright 2009 Elsevier B.V., All rights reserved.
- [169] Michael Procopio, Jane Mulligan, and Greg Grudic. Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments. *Journal of Field Robotics*, 26:145 – 175, 02 2009.
- [170] Shengyan Zhou, Junqiang Xi, Matthew W. McDaniel, Takayuki Nishihata, Phil Salesses, and Karl Iagnemma. Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain. *J. Field Robotics*, 29:277–297, 2012.
- [171] Andres Milioto, Philipp Lottes, and Cyrill Stachniss. Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2229–2235. IEEE, 2018.
- [172] Tianrui Guan, Divya Kothandaraman, Rohan Chandra, Adarsh Jagan Sathyamoorthy, Kasun Weerakoon, and Dinesh Manocha. Ga-nav: Efficient terrain segmentation for robot navigation in unstructured outdoor environments. *IEEE Robotics and Automation Letters*, 7(3):8138–8145, 2022.
- [173] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Proceedings of 11th International Conference on Field and Service Robotics (FSR '17)*, pages 335 – 350, September 2017.
- [174] Alberto Lacaze, Edward Mottern, and Bryan Brilhart. Off-road autonomous mobility. In *2019 NDIA Ground Vehicle Systems Engineering and Technology Symposium*, 2019.
- [175] Adam Plevoy, Craig Knuth, Katie M. Popek, and Kapil D. Katyal. Complex terrain navigation via model error prediction. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9411–9417, 2022.
- [176] Kasun Weerakoon, Adarsh Jagan Sathyamoorthy, Jing Liang, Tianrui Guan, Utsav Patel, and Dinesh Manocha. GrASPE: Graph based Multimodal Fusion for Robot Navigation in Unstructured Outdoor Environments. *arXiv e-prints*, page arXiv:2209.05722, September 2022.
- [177] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.
- [178] Marcel Häselich, Marc Arends, Nicolai Wojke, Frank Neuhaus, and Dietrich Paulus. Probabilistic terrain classification in unstructured environments. *Robotics and Autonomous Systems*, 61:1051–1059, 10 2013.

- [179] Sebastian Pütz, Thomas Wiemann, Jochen Sprickerhof, and Joachim Hertzberg. 3d navigation mesh generation for path planning in uneven terrain. *IFAC-PapersOnLine*, 49(15):212–217, 2016. 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016.
- [180] Maggie Wigness, Sungmin Eum, John G Rogers, David Han, and Heesung Kwon. A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- [181] Peng Jiang, Philip Osteen, Maggie Wigness, and Srikanth Saripalli. Rellis-3d dataset: Data, benchmarks and analysis, 2020.
- [182] Lorenz Wellhausen, René Ranftl, and Marco Hutter. Safe robot navigation via multi-modal anomaly detection. *IEEE Robotics and Automation Letters*, 5(2):1326–1333, 2020.
- [183] Kai Zhao, Mingming Dong, and Liang Gu. A new terrain classification framework using proprioceptive sensors for mobile robots. *Mathematical Problems in Engineering*, 2017:1–14, 09 2017.
- [184] Mohammed Abdessamad Bekhti, Yuichi Kobayashi, and Kazuki Matsumura. Terrain traversability analysis using multi-sensor data correlation by a mobile robot. In *2014 IEEE/SICE International Symposium on System Integration*, pages 615–620, 2014.
- [185] Boris Sofman, Ellie Lin Ratliff, J. Andrew (Drew) Bagnell, Nicolas Vandapel, and Anthony (Tony) Stentz. Improving robot navigation through self-supervised online learning. In *Proceedings of Robotics: Science and Systems (RSS '06)*, August 2006.
- [186] Mirko Nava, Jérôme Guzzi, R. Omar Chavez-Garcia, Luca M. Gambardella, and Alessandro Giusti. Learning long-range perception using self-supervision from short-range sensors and odometry. *IEEE Robotics and Automation Letters*, 4(2):1279–1286, 2019.
- [187] Larry Matthies, Michael Turmon, Andrew Howard, Anelia Angelova, Benyang Tang, and Eric Mjolsness. Learning for autonomous navigation: Extrapolating from underfoot to the far field. 01 2005.
- [188] Max Bajracharya, Andrew Howard, Larry H. Matthies, Benyang Tang, and Michael Turmon. Autonomous off-road navigation with end-to-end learning for the lagr program. *J. Field Robot.*, 26(1):3–25, jan 2009.
- [189] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Miller, and Yann LeCun. A multirange architecture for collision-free off-road robot navigation. *J. Field Robot.*, 26(1):52–87, jan 2009.
- [190] Kurt Konolige, Motilal Agrawal, Morten Rufus Blas, Robert C. Bolles, Brian Gerkey, Joan Solà, and Aravind Sundaresan. Mapping, navigation, and learning for off-road traversal. *J. Field Robot.*, 26(1):88–113, jan 2009.

- [191] A. Angelova, L. Matthies, D. Helmick, G. Sibley, and P. Perona. Learning to predict slip for ground robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3324–3331, 2006.
- [192] Anelia Angelova, Larry H. Matthies, Daniel M. Helmick, and Pietro Perona. Slip prediction using visual information. In *Robotics: Science and Systems*, 2006.
- [193] Camilo Ordonez, Ryan Alicea, Brandon Rothrock, Kyle Ladyko, Jeremy Nash, Rohan Thakker, Shreyansh Daftry, Mario Harper, Emmanuel Collins, and Larry Matthies. Characterization and traversal of pliable vegetation for robot navigation. 11 2018.
- [194] S.L. Laubach, J. Burdick, and L. Matthies. An autonomous path planner implemented on the rocky 7 prototype microrover. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 1, pages 292–297 vol.1, 1998.
- [195] David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010.
- [196] Konstantin Zakharov, Anton Saveliev, and Oleg Sivchenko. Energy-efficient path planning algorithm on three-dimensional large-scale terrain maps for mobile robots. In *International Conference on Interactive Collaborative Robotics*, pages 319–330. Springer, 2020.
- [197] Sriram Siva, Maggie Wigness, John G. Rogers, and Hao Zhang. Robot adaptation for generating consistent navigational behaviors over unstructured off-road terrain, 2021.
- [198] David Bradley, Scott Thayer, Anthony (Tony) Stentz, and Peter Rander. Vegetation detection for mobile robot navigation. Technical Report CMU-RI-TR-04-12, Carnegie Mellon University, Pittsburgh, PA, February 2004.
- [199] Duong-Van Nguyen, Lars Kuhnert, and Klaus Kuhnert. Structure overview of vegetation detection. a novel approach for efficient vegetation detection using an active lighting system. *Robotics and Autonomous Systems*, 60:498–508, 04 2012.
- [200] D.-V. Nguyen, L. Kuhnert, S. Thamke, J. Schlemper, and K.-D. Kuhnert. A novel approach for a double-check of passable vegetation detection in autonomous ground vehicles. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 230–236, 2012.
- [201] A. Talukder, R. Manduchi, R. Castano, K. Owens, L. Matthies, A. Castano, and R. Hogg. Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 708–713 vol.1, 2002.
- [202] Shreya Lohar, Lei Zhu, Stanley Young, Peter Graf, and Michael Blanton. Sensing technology survey for obstacle detection in vegetation. *Future Transportation*, 1(3):672–685, 2021.

- [203] Camilo Ordonez, Nikhil Gupta, Oscar Chuy, and Emmanuel G. Collins. Momentum based traversal of mobility challenges for autonomous ground vehicles. In *2013 IEEE International Conference on Robotics and Automation*, pages 752–759, 2013.
- [204] Ryan Luis Alicea. Efficient control and locomotion strategies in unstructured, natural environments: A study of vegetation-rich and fluid-covered terrain. 2019.
- [205] Camilo Ordonez, Ryan Alicea, Brandon Rothrock, Kyle Ladyko, Mario Harper, Sisir Karumanchi, Larry Matthies, and Emmanuel Collins. Modeling and traversal of pliable materials for tracked robot navigation. In Robert E. Karlson, Douglas W. Gage, Charles M. Shoemaker, and Hoa G. Nguyen, editors, *Unmanned Systems Technology XX*, volume 10640, page 106400F. International Society for Optics and Photonics, SPIE, 2018.
- [206] Kai Wurm, Henrik Kretzschmar, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. Identifying vegetation from laser data in structured outdoor environments. *Robotics and Autonomous Systems*, 62:675–684, 05 2014.
- [207] Kai M Wurm, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. Improving robot navigation in structured outdoor environments by identifying vegetation from laser data. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1217–1222. IEEE, 2009.
- [208] Pietro Astolfi, Alessandro Gabrielli, Luca Bascetta, and Matteo Matteucci. Vineyard autonomous navigation in the echord++ grape experiment. 51:704–709, 01 2018.
- [209] Dominik Belter, Jan Wietrzykowski, and Piotr Skrzypczyński. Employing natural terrain semantics in motion planning for a multi-legged robot. *Journal of Intelligent and Robotic Systems*, 93, 03 2019.
- [210] Timothy Overbye and Srikanth Saripalli. Fast local planning and mapping in unknown off-road terrain. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5912–5918, 2020.
- [211] Timothy Overbye and Srikanth Saripalli. Path optimization for ground vehicles in off-road terrain. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7708–7714, 2021.
- [212] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020.
- [213] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.
- [214] Timon Homberger, Lorenz Wellhausen, Péter Fankhauser, and Marco Hutter. Support surface estimation for legged robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8470–8476, 2019.

- [215] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.
- [216] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, pages 461–464, 1978.
- [217] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, 1991.
- [218] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. In *16th European Conference Computer Vision (ECCV 2020)*, August 2020.
- [219] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.
- [220] Tianyi Wu, Sheng Tang, Rui Zhang, and Yongdong Zhang. CGNet: A Lightweight Context Guided Network for Semantic Segmentation. *arXiv e-prints*, page arXiv:1811.08201, November 2018.
- [221] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.
- [222] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.
- [223] Maggie Wigness, Sungmin Eum, John G Rogers, David Han, and Heesung Kwon. A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- [224] Peng Jiang, Philip Osteen, Maggie Wigness, and Srikanth Saripalli. Rellis-3d dataset: Data, benchmarks and analysis, 2020.
- [225] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [226] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

- [227] Masaki Kuribayashi, Seita Kayukawa, Jayakorn Vongkulbhisal, Chieko Asakawa, Daisuke Sato, Hironobu Takagi, and Shigeo Morishima. Corridor-walker: Mobile indoor walking assistance for blind people to avoid obstacles and recognize intersections. *Proceedings of the ACM on Human-Computer Interaction*, 6(MHCI):1–22, 2022.
- [228] Minjie Hua, Yibing Nan, and Shiguo Lian. Small obstacle avoidance based on rgb-d semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019.
- [229] Yutian Wu, Yueyu Wang, Shuwei Zhang, and Harutoshi Ogai. Deep 3d object detection networks using lidar data: A review. *IEEE Sensors Journal*, 21(2):1152–1171, 2020.
- [230] Brian P Gerkey and Kurt Konolige. Planning and control in unstructured terrain. In *ICRA workshop on path planning on costmaps*. Citeseer, 2008.
- [231] Xiaowen Ying and Mooi Choo Chuah. Uctnet: Uncertainty-aware cross-modal transformer network for indoor rgb-d semantic segmentation. In *European Conference on Computer Vision*, pages 20–37. Springer, 2022.
- [232] Tianrui Guan, Divya Kothandaraman, Rohan Chandra, Adarsh Jagan Sathyamoorthy, Kasun Weerakoon, and Dinesh Manocha. Ga-nav: Efficient terrain segmentation for robot navigation in unstructured outdoor environments. *IEEE Robotics and Automation Letters*, 7(3):8138–8145, 2022.
- [233] Taylor M Petty, Juan D Fernandez, Jason N Fischell, and Luis A De Jesús-Díaz. Lidar attenuation through a physical model of grass-like vegetation. *Journal of Autonomous Vehicles and Systems*, 2(2):021003, 2022.
- [234] David V. Lu and William D. Smart. Towards more efficient navigation for robots and humans. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1707–1713, 2013.
- [235] Jérôme Guzzi, Alessandro Giusti, Luca M. Gambardella, Guy Theraulaz, and Gianni A. Di Caro. Human-friendly robot navigation in dynamic environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 423–430, 2013.
- [236] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, and et al. Language Models are Few-Shot Learners. *arXiv e-prints*, page arXiv:2005.14165, May 2020.
- [237] OpenAI. Gpt-4 technical report. 2023.
- [238] Meta AI. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv e-prints*, page arXiv:2307.09288, July 2023.
- [239] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved Baselines with Visual Instruction Tuning. *arXiv e-prints*, page arXiv:2310.03744, October 2023.
- [240] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models. *arXiv e-prints*, page arXiv:2312.11805, December 2023.

- [241] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. *arXiv e-prints*, page arXiv:2304.08485, April 2023.
- [242] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. *arXiv e-prints*, page arXiv:2103.00020, February 2021.
- [243] Nur Muhammad Mahi Shafiullah, Chris Paxton, Lerrel Pinto, Soumith Chintala, and Arthur Szlam. Clip-fields: Weakly supervised semantic fields for robotic memory. *arXiv preprint arXiv: Arxiv-2210.05663*, 2022.
- [244] Huy Ha and Shuran Song. Semantic abstraction: Open-world 3d scene understanding from 2d vision-language models. In *6th Annual Conference on Robot Learning*, 2022.
- [245] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, and et al. PIVOT: Iterative Visual Prompting Elicits Actionable Knowledge for VLMs. *arXiv e-prints*, page arXiv:2402.07872, February 2024.
- [246] Vishnu Sashank Dorbala, Gunnar A Sigurdsson, Jesse Thomason, Robinson Piramuthu, and Gaurav S. Sukhatme. CLIP-nav: Using CLIP for zero-shot vision-and-language navigation. In *Workshop on Language and Robotics at CoRL 2022*, 2022.
- [247] Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. LM-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *6th Annual Conference on Robot Learning*, 2022.
- [248] Peiqi Liu, Yaswanth Orru, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*, 2024.
- [249] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, and et al. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.
- [250] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 2023.
- [251] Adarsh Jagan Sathyamoorthy, Kasun Weerakoon, Tianrui Guan, Mason Russell, Damon Conover, Jason Pusey, and Dinesh Manocha. Vern: Vegetation-aware robot navigation in dense unstructured outdoor environments. *arXiv e-prints*, pages arXiv–2303, 2023.
- [252] Luca Di Giammarino, Irvin Aloise, Cyrill Stachniss, and Giorgio Grisetti. Visual place recognition using lidar intensity information. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4382–4389. IEEE, 2021.

- [253] Keith Sullivan, Wallace Lawson, and Donald Sofge. Fusing laser reflectance and image data for terrain classification for small autonomous robots. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1656–1661. IEEE, 2014.
- [254] Stefan Laible, Yasir Niaz Khan, Karsten Bohlmann, and Andreas Zell. 3d lidar-and camera-based terrain classification under different lighting conditions. In *Autonomous Mobile Systems 2012: 22. Fachgespräch Stuttgart, 26. bis 28. September 2012*, pages 21–29. Springer, 2012.
- [255] Christophe Reymann and Simon Lacroix. Improving lidar point cloud classification using intensities and multiple echoes. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5122–5128. IEEE, 2015.
- [256] Jiadong Guo, Paulo VK Borges, Chanoh Park, and Abel Gawel. Local descriptor for robust place recognition using lidar intensity. *IEEE Robotics and Automation Letters*, 4(2):1470–1477, 2019.
- [257] Mohammad Aldibaja, Noaki Suganuma, Reo Yanase, and Keisuke Yoneda. Reliable graph-slam framework to generate 2d lidar intensity maps for autonomous vehicles. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–6, 2020.
- [258] Chongyang Wei, Tao Wu, and Hao Fu. Estimating initial guess of localization by line matching in lidar intensity maps. In *Information Technology and Intelligent Transportation Systems: Volume 1, Proceedings of the 2015 International Conference on Information Technology and Intelligent Transportation Systems ITITS 2015, held December 12-13, 2015, Xi'an China*, pages 577–588. Springer, 2017.
- [259] Ioan Andrei Barsan, Shenlong Wang, Andrei Pokrovsky, and Raquel Urtasun. Learning to localize using a lidar intensity map. *arXiv preprint arXiv:2012.10902*, 2020.
- [260] Timothy D Barfoot, Colin McManus, Sean Anderson, Hang Dong, Erik Beerepoot, Chi Hay Tong, Paul Furgale, Jonathan D Gammell, and John Enright. Into darkness: Visual navigation based on a lidar-intensity-image pipeline. In *Robotics Research: The 16th International Symposium ISRR*, pages 487–504. Springer, 2016.
- [261] Young Ha Shin and Dong-Cheon Lee. True orthoimage generation using airborne lidar data with generative adversarial network-based deep learning model. *Journal of Sensors*, 2021:1–25, 2021.
- [262] Franck Garestier, Patrice Bretel, Olivier Monfort, Franck Levoy, and Emilie Poullain. Anisotropic surface detection over coastal environment using near-ir lidar intensity maps. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(2):727–739, 2014.
- [263] Julien Dupeyroux, Stéphane Viollet, and Julien R Serres. An ant-inspired celestial compass applied to autonomous outdoor robot navigation. *Robotics and Autonomous Systems*, 117:40–56, 2019.

- [264] Jing Li, Hui Qin, Junzheng Wang, and Jiehao Li. Openstreetmap-based autonomous navigation for the four wheel-legged robot via 3d-lidar and ccd camera. *IEEE Transactions on Industrial Electronics*, 69(3):2708–2717, 2021.
- [265] Goran Huskić, Sebastian Buck, and Andreas Zell. Gerona: generic robot navigation: a modular framework for robot navigation and control. *Journal of Intelligent & Robotic Systems*, 95(2):419–442, 2019.
- [266] Kasun Weerakoon, Adarsh Jagan Sathyamoorthy, Jing Liang, Tianrui Guan, Utsav Patel, and Dinesh Manocha. Graspe: Graph based multimodal fusion for robot navigation in outdoor environments. *IEEE Robotics and Automation Letters*, 2023.
- [267] Adarsh Jagan Sathyamoorthy, Kasun Weerakoon, Tianrui Guan, Jing Liang, and Dinesh Manocha. Terrapn: Unstructured terrain navigation using online self-supervised learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7197–7204, 2022.
- [268] Timon Homberger, Lorenz Wellhausen, Péter Fankhauser, and Marco Hutter. Support surface estimation for legged robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8470–8476. IEEE, 2019.
- [269] Mohamed Elnoor, Adarsh Jagan Sathyamoorthy, Kasun Weerakoon, and Dinesh Manocha. Pronav: Proprioceptive traversability estimation for autonomous legged robot navigation in outdoor environments. *arXiv preprint arXiv:2307.09754*, 2023.
- [270] Dmitriy Rivkin, Gregory Dudek, Nikhil Kakodkar, David Meger, Oliver Limoyo, Michael Jenkin, Xue Liu, and Francois Hogan. Ansel photobot: A robot event photographer with semantic intelligence. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8262–8268, 2023.
- [271] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2023.
- [272] Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, brian ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *7th Annual Conference on Robot Learning*, 2023.
- [273] Peihao Chen, Xinyu Sun, Hongyan Zhi, Runhao Zeng, Thomas H. Li, Gaowen Liu, Mingkui Tan, and Chuang Gan. A²Nav: Action-Aware Zero-Shot Robot Navigation by Exploiting Vision-and-Language Ability of Foundation Models. *arXiv e-prints*, page arXiv:2308.07997, August 2023.
- [274] Peiyuan Zhu, Lotfi El Hafi, and Tadahiro Taniguchi. Visual-language decision system through integration of foundation models for service robot navigation. In *2024 IEEE/SICE International Symposium on System Integration (SII)*, pages 1288–1295. IEEE, 2024.

- [275] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10608–10615, 2023.
- [276] Raffaele Soloperto, Johannes Köhler, Frank Allgöwer, and Matthias A. Müller. Collision avoidance for uncertain nonlinear systems with moving obstacles using robust model predictive control. In *2019 18th European Control Conference (ECC)*, pages 811–817, 2019.
- [277] James Guthrie, Marin Kobilarov, and Enrique Mallada. Closed-form minkowski sum approximations for efficient optimization-based collision avoidance. In *2022 American Control Conference (ACC)*, pages 3857–3864, 2022.
- [278] Lasitha Weerakoon, Gurtajbir Singh Herr, Jasmine Blunt, Miao Yu, and Nikhil Chopra. Cartographer_glass: 2d graph slam framework using lidar for glass environments. *arXiv preprint arXiv:2212.08633*, 2022.
- [279] Xun Wang and JianGuo Wang. Detecting glass in simultaneous localisation and mapping. *Robotics and Autonomous Systems*, 88:97–103, 2017.
- [280] Fangyu Liu, Guy Emerson, and Nigel Collier. Visual spatial reasoning. *Transactions of the Association for Computational Linguistics*, 11:635–651, 2023.
- [281] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.