ABSTRACT

Title of Dissertation: ON DATA-BASED MAPPING AND NAVIGATION OF UNMANNED GROUND VEHICLES

Gurtajbir Singh Herr
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Nikhil Chopra
Department of Mechanical Engineering

Unmanned ground vehicles (UGVs) have seen tremendous advancement in their capabilities and applications in the past two decades. With several key algorithmic and hardware breakthroughs and advancements in deep learning, UGVs are quickly becoming ubiquitous (finding applications as self-driving cars, for remote site inspections, in hospitals and shopping malls, among several others). Motivated by their large-scale adoption, this dissertation aims to enable the navigation of UGVs in complex environments. In this dissertation, a supervised learning-based navigation algorithm that utilizes model predictive control (MPC) for providing training data is developed. Improving MPC performance by data-based modelling of complex vehicle dynamics is then addressed. Finally, this dissertation deals with detecting and registering transparent objects that may deteriorate navigation performance.

Navigation in dynamic environments poses unique challenges, particularly due to the limited knowledge of the decisions made by other agents and their objectives. In this dissertation, a solution that utilizes an MPC-based planner as an *expert* to generate high-quality

motion commands for a car-like robot operating in a simulated dynamic environment is proposed. These commands are then used to train a deep neural network, which learns to navigate. The deep learning-based planner is further enhanced with safety margins to improve its effectiveness in collision avoidance. The performance of the proposed method through simulations and real-world experiments, demonstrating its superiority in terms of obstacle avoidance and successful mission completion is showcased. This research has practical implications for the development of safer and more efficient autonomous vehicles.

Many real-world applications rely on MPC to control UGVs due to its safety guarantees and constraint satisfaction properties. However, the performance of such MPC-based solutions is heavily reliant on the accuracy of the motion model. This dissertation addresses this challenge by exploring a data-based approach to discovering vehicle dynamics. Unlike existing physics-based models that require extensive testing setups and manual tuning for new platforms and driving surfaces, our approach leverages the universal differential equations (UDEs) framework to identify unknown dynamics from vehicle data. This innovative approach, which does not make assumptions about the unknown dynamics terms and directly models the vector field, is then deployed to showcase its efficacy. This research opens up new possibilities for more accurate and adaptable motion models for UGVs.

With the increasing adoption of glass and other transparent materials, UGVS must be able to detect and register them for reliable navigation. Unfortunately, such objects are not easily detected by LiDARs and cameras. In this dissertation, algorithms for detecting and including glass objects in a Graph SLAM framework were studied. A simple and computationally inexpensive glass detection scheme to detect glass objects is utilized. The methodology to incorporate the identified objects into the occupancy grid maintained by such a framework is the

presented. The issue of *drift accumulation* that can affect mapping performance when operating in large environments is also addressed.

ON DATA-BASED MAPPING AND NAVIGATION OF UNMANNED GROUND VEHICLES

by

Gurtajbir Singh Herr

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:
Professor Nikhil Chopra, Chair/Advisor
Professor Balakumar Balachandran
Professor Miao Yu
Assistant Professor Yancy Diaz-Mercado
Associate Professor Anubhav Datta, Dean's Representative

UNIVERSITY OF MARYLAND

2181 Glenn L. Martin Hall
College Park, MD 20742-3035
TEL: 301-405-2410
http://www.enme.umd.edu

April 12, 2024

Re: Previously Published Materials appearing in Thesis or Dissertation

Dean of the Graduate School,

Gurtajbir Singh Herr has:

☐ NOT INCLUDED any previously published works within their thesis or dissertation.

☒ INCLUDED one or more previously published works within their thesis or dissertation. This letter certifies that the examining committee for the student has determined that the student made a substantial contribution to the previously published work. The inclusion of the previously published work has the approval of the thesis or dissertation advisor and the Graduate Director.

Sincerely,

Peter Sandborn
Director of Graduate Studies
Department of Mechanical Engineering
University of Maryland

Sincerely,

Nikhil Chopra
Advisor for Gurtajbir Singh Herr

Dedication

To my parents, and the memory of my grand parents and Chachaji

Acknowledgments

I want to express my deepest gratitude to my doctoral advisor, Dr. Nikhil Chopra, for his guidance and mentorship. His unwavering support has been instrumental in shaping my academic journey. His passion for exploring new and courageous ideas is exemplary, and his compassionate approach certainly made my PhD journey a rewarding and enjoyable experience.

I would like to extend my sincerest gratitude to my dissertation committee members Dr. Balakumar Balachandran, Dr. Miao Yu, Dr. Yancy Diaz-Mercado and Dr. Anubhav Datta for agreeing to serve, and providing insightful perspectives. I would like to convey my thanks to all my teachers at UMD with whom I took courses and had the oppurtunity to serve as a TA. Big thanks to everyone in the Mechanical Engineering office for being super helpful and always prioritizing the needs of the students.

I am thankful for meeting a lot of wonderful people here at UMD. I would like to especially acknowledge Lasitha Weerakoon and Nirupam Gupta, for their encouragement and intellectual exchange throughout this journey. Their support and friendship have made the research process more enjoyable and rewarding.

To my dear parents, your guidance, sacrifices, and unwavering belief in my abilities have been the guiding force behind my academic pursuits. I am forever grateful for your endless love and for instilling in me the values of perseverance and dedication. To my cherished wife, your patience, understanding, and unwavering support have been my rock throughout this endeavor.

Your belief in my dreams and sacrifices made on my behalf have been the driving force propelling me forward. To my beloved sister, you have have always had my back and no words can express my love and gratitude.

Finally, I also dedicate this dissertation to my beloved son, Sohrab. May it serve as a token of my gratitude for the endless joy and motivation you have brought into my life. I am forever grateful for the privilege of being your parent and for the opportunity to share this milestone with you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BFS	Breadth First Search
CNN	Convolutional Neural Network
DFS	Depth First Search
DL	Deep Learning
DRL	Deep Reinforcement Learning
DWA	Dynamic Window Approach
GP	Gaussian Process
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LMPC	Learning Model Predictive Controller
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
MSE	Mean Squared Error
NODE	Neural Ordinary Differential Equations
ORCA	Optimal Reciprocal Collision Avoidance
PFM	Potential Field Method
PGO	Pose Graph Optimization
PRM	Probabilistic Road Map
RRT	Rapidly Exploring Random Trees
RVO	Reciprocal Velocity Obstacle
VOM	Velocity Obstacle Method
SLAM	Simultaneous Localization and Mapping
TEB	Timed Elastic Band
UDE	Universal Differential Equation
UGV	Unmanned Ground Vehicle

Chapter 1: Introduction

This dissertation addresses the navigation of UGVs in complex environments. As such, it is important to introduce the relevant concepts here. In this dissertation, a supervised learning-based navigation algorithm that utilizes model predictive control (MPC) for providing training data is developed. Therefore, in this chapter, autonomous navigation, in both, its classical flavor and the more recent machine learning based approaches is addressed. This dissertation also addresses data-based modelling of complex vehicle dynamics. Thus, an overview of data-based system identification for UGV dynamics that have been proposed in the recent literature is provided. This dissertation delves into mapping in environments that contain transparent objects. Hence, a brief overview of mapping with UGVs is provided. This followed by identifying key challenges and research opportunities that emerge. Next, the key contributions of this dissertation are presented.

1.1 Autonomous navigation

Autonomous navigation is a key enabler in the development of modern robotic systems that can operate without human intervention. It constitutes the capability of a robot to move to a desired goal smoothly while avoiding collisions [1]. Traditionally, roboticists have dealt with this problem in a hierarchical fashion. A coarse plan to the goal is prepared by a *global planner* based frequently on a pre-existing map of the environment while current perceptual information

may also be used to update this representation from time to time. This task is then divided into smaller sub-tasks which are handled by a *local planner*. The local planner issues low level motion commands to steer the robot to these intermediate targets [2]. In this dissertation, the focus is on the motion control via a deep learning (DL) based local planner rather than the higher level global planning problem.

1.1.1 Classical approach to autonomous navigation

Classically, the autonomous navigation problem has been dealt with the *sense, plan and act* approach wherein sensor data is first used to create a situational understanding or a map of the environment. The global planner then carries out a search for a feasible path on such a representation. Popular search based methods commonly used are Dijkstra's algorithm [3], depth first search (DFS) [4], breadth first search (BFS) [4] and A* [5], among others. Sampling based methods such rapidly exploring random trees (RRT) [6] and probabilistic road maps (PRM)[7] are more popular for large-scale problems and continuous workspaces.

There exists a large body of literature for the motion control of the robot in order to navigate it to the sub-goals or follow sections derived from the global plan. Potential field method (PFM) [8] is one of the earliest reaction based algorithms that address this problem. Similar to a ball rolling downhill, this method is based on the robot following the gradient of an artificial potential field. The goal is designed to be a minimum and hence, pulls the robot towards itself by applying an attractive force. The obstacles are modeled as peaks so that they repel the robot away from themselves. If new obstacles appear in the vicinity of the robot, the potential field has to be updated to accommodate them. In this approach, the robot is assumed to be a point and the

orientation of the robot is usually ignored. The shortcomings of this approach include getting stuck in local minima and inability to go through narrow passages. Some of the shortcomings of the PFM were addressed in the extended potential field [9] and harmonic potential field [10] methods.

The dynamic window approach (DWA) [11] is a sampling based optimization framework used frequently for local planning. The control actions are sampled from a *feasible velocity* space and the trajectory is rolled out using the kinematics model of the robot [12]. Control actions are kept constant in the prediction horizon and the resulting trajectory is evaluated on the basis of a cost function and the robot's kinodynamic constraints. The cost function may take into account the robot's progress towards the goal, the magnitude of its velocity and its distance from obstacles [13]. This method does not require the cost function to be continuous and can be used with a grid based framework such as an occupancy grid map. One short coming of this approach apart from being sub-optimal is that it is suited for differential drive and omni-directional robots only but not for car-like robots since the latter can not change orientation without translating.

Among the classical approaches, a very popular technique is the velocity obstacle method (VOM) first introduced in [14]. This method like the DWA, can handle both static and dynamic obstacles. In this method, robot velocities are sampled from outside the *velocity obstacles*, a set of robot velocities that would result in a collision with an obstacle based on its current velocity at a point in the future. These sampled velocities are chosen from an intersection with the admissible velocities defined by the robot's acceleration constraints. However, in dense and dynamic environments, collisions with obstacles occur. The reciprocal velocity obstacle (RVO) method introduced in [15] builds upon the VOM and applies it to a multi-agent setting. They factor in the reactive behavior of other agents by assuming that the other agents also employ a

similar collision avoidance strategy. Building upon the work in [15], authors derive sufficient conditions for collision free motion among n agents in a multi-agent setting [16]. This approach, named by the authors as optimal reciprocal collision avoidance (ORCA), is widely regarded as the state-of-the-art among the classical or non-learning based methods.

Another recent technique that is modelled as a model predictive control (MPC) problem is the *Timed-Elastic-Band* (TEB) approach [17]. This method incorporates the temporal information into the cost function, encodes the kinodynamic constraints of the robot as the constraints of the optimization problem and looks for a solution that results in minimum time to goal. The trajectory planning and controls is formulated as a sparse optimization problem that is solved using an efficient non-linear optimization solver [18]. An advantage of this approach is that it is also applicable to car-like robots [19].

Despite a tremendous amount of engineering effort and several successful applications, a major drawback of the classical paradigm is that the performance is often inferior to human teleoperation and these approaches are not robust in highly constrained and diverse environments [1].

1.1.2 Machine learning based autonomous navigation

The rise of deep learning (DL) [20], the ability to collect large datasets in real world scenarios [21] and simulations [22, 23] along with advancement in computing hardware, has resulted in an increased interest in ML based autonomous navigation. Learning based solutions aim to obviate the need for time consuming hand tuning, which is a major short coming of the traditional autonomous navigation systems. They also strive to make these systems more

generally applicable for out-of-the-box deployment. In [24], the authors trained a DL model in a supervised fashion. The data was generated in simulations where the robot was given a fixed goal and the path planning and navigation was done using Dijkstra's algorithm [3] and the DWA planner, respectively. The model consisted of a 1D Convolutional Neural Network (CNN) followed by fully connected layers to output the translational and rotational velocities. This method demonstrated successful navigation in simulation and real world settings. Another class of methods namely, Inverse Reinforcement Learning, aim to estimate the cost functions of other dynamic agents in the environment and use these to achieve socially compliant navigation [25, 26]. In contrast to supervised learning and inverse reinforcement learning, reinforcement learning (RL) methods are more 'self dependent' in that they proceed by exploring the state space in a simulated environment without supervision [27]. Reinforcement learning [28] is a sequential decision making paradigm which is typically modeled as a Markov Decision Process (MDP). This consists of a state space, an action space and a state transition model. The aim of the RL framework is to find a decision making policy which maps the the state of the system to an action. In the context of RL for autonomous navigation, the most popular policy learning algorithms used are the asynchronous actor-critic method [29] and proximal policy optimization [30].

ML based approaches to autonomous navigation have seen both end-to-end [21][31] as well as subsystem level [24] techniques proposed. The subsystem here refers to either the global or the local planner in the navigation stack. Some research has also focused on automated parameter selection for these navigation subsystems [32]. A large body of work in ML based autonomous navigation falls under end-to-end methods [24, 31]. The end-to-end approach aims to replace the entire navigation stack with a learned model. This approach has the benefit that it abstracts away

the internal workings of the navigation stack and deals directly with learning the mapping from perceptual inputs to the necessary control actions. These methods have been used to accomplish both fixed-goal as well as moving goal navigation. In the moving goal case, the robot constantly moves forward as if in the pursuit of a moving virtual goal in front of it [1]. The subsystem level approaches focus on harnessing the power of learning based solutions for a subsystem of the navigation stack. More commonly these methods are intended to replace the local planner with some works like [33] and [34] focussing on replacing the conventional global planner and providing sub-goals to the local planner respectively. Some prominent recent examples of DL models replacing the local planner are the attention mechanisms based local planner presented in [35], an imitation learning based model called Intention-Net proposed in [36] and the Learning from Hallucination paradigm introduced in [37]. In an extensive study of end-to-end methods for autonomous navigation in [38], the authors noted that for application to unseen environments an end-to-end approach that replaced a map-based global planner was not advisable.

1.2 Machine learning based identification of UGV dynamics

When wheeled robots operate at high speeds and accelerations, the importance of an accurate dynamics model becomes paramount. The simpler kinematic models that suffice at lower speeds are not sufficient in these regimes. This is especially true in applications such as autonomous racing where the vehicles operate at limits of their capabilities [39]. An inaccurate model might lead to degradation in controller performance and may prove catastrophic. Traditional system identification methods for modelling vehicle dynamics rely on hard to obtain physics-based models. This is primarily due to the difficulty in modelling complex nonlinear

phenomenon such as those at work in modelling of lateral tire forces which cause lateral tire slip [40]. This may require extensive testing setups in order to identify corresponding system parameters which has to be repeated for each new platform and also for different surfaces.

There have been several data-based methods that have been proposed for this task. The Learning Model Predictive Controller(LMPC) proposed in [41] learns from the data recorded during each lap around the racetrack to improve an affine time-varying model. The model learning strategy in the aforementioned work is improved upon in [42] by utilizing a nominal model and learning the error dynamics with collected data. The use of Gaussian Process (GP) models for modelling vehicle dynamics was proposed in [43]. This was further developed in [40] to build upon an extended kinematic model using the residual physics which is captured by a GP model. A more comprehensive study of GP models for this problem that takes into consideration the kernel choice, sample sample size and different racing conditions is presented in [39]. The use of ANNs for learning forward kinodynamics for high speed robot control was proposed in [44] where a 6 layered ANN with 256 units per layer was used.

The advent of Neural Ordinary Differential Equations (NODEs) [45] paved the way for exploiting the expressive power of neural networks for dynamical system modelling. Since neural networks are universal approximators [46], they present an attractive avenue for modelling complex system dynamics which are difficult to obtain from physics-based modelling. In this framework, the vector field is modeled using a neural network and the system evolution can be predicted using a differential equation solver. However, NODEs can have a high sample complexity and thus require a large amounts of training data.

1.3 Mapping of indoor environments by UGVs

Autonomous robots require a representation of the world around them in order to make sense of their tasks. In case the localization estimate is available to the robot, mapping is relatively easy. The problem reduces to a scenario which is popularly known as *mapping with known poses*. This is , however, rarely the case in indoor environments. External localization estimates such as those from the Global Positioning System (GPS) are unavailable indoors. In the absence of such an estimate, it becomes much harder for the robot to carry out the mapping task. Here is where Simultaneous Localization and Mapping [47], popularly known as SLAM, comes into picture. As the name suggests, these techniques carry out the task of location estimation and mapping simultaneously. Broadly, there are three main categories of approaches to SLAM namely, extended Kalman filter (EKF) based methods [48], particle filter [49], and optimization based methods [50](also known as Graph SLAM). Among these methods, Graph SLAM based techniques are widely considered to be state-of-the-art [51]. A popular method for creating representations of the environment are occupancy grid maps [52]. As the name suggests, occupancy grid maps discretize the world or the environment into cells. The area defined by each cell is either completely free or occupied and this is modeled by a binary random variable variable which takes on the values 0 or 1. The choice for the representation suitable to an application depends on factors such as the level of detail required and compatibility with downstream systems such as planning and navigation.

1.4 Challenges and opportunities

Autonomous indoor navigation of UGVs is an area of active research interest. As the ubiquity of these robots and their functional role in human lives continues to grow, it becomes increasingly important that their navigation be made more efficient and safe. Non-learning based methods often have simplifying assumptions regarding motion models and fail to model the intent of non-communicating, independent decision making agents [27]. Learning based methods, especially deep reinforcement learning (DRL), have shown promise in dealing with these shortcomings. At the same time, DRL based methods tend to be very data hungry and the sim-to-real transfer capability requires high fidelity simulation environments.

MPC based solutions are used extensively in controlling UGVs. The performance of these methods depends strongly of the quality of the dynamics model. Traditional system identification approaches are time and resource consuming, therefore the use of machine learning based methods looks promising.

The two most popular sensing modalities for SLAM systems for UGVs used today are lidars and cameras. The increased use of glass and other transparent surfaces inside modern buildings presents a considerable challenge for these sensors as they are not reliably detectable from most viewpoints. This is problematic from the point of view of mapping as well as navigation where it can become a safety issue.

1.5 Contributions

This dissertation aims to address the challenges laid out in Section 1.4. Specifically, for navigation in dynamic environments, this dissertation leverages an optimization based *expert* to generate high quality labels in simulated dynamic environments. The data generation process is automated by carefully designing simulations such that the burden associated with manually generated labels (which require a human in the loop) is obviated. The trained model is then tested on unseen environments in both simulation and real world experiments. The learned model outperforms the open source adaptations of the TEB local planner suitable for dynamic environments.

A universal differential equations (UDE) based identification approach is presented. This approach does not make any assumptions about the structure of the unknown dynamics terms and directly models the vector field. This enables the learnt model to be used at different timescales and is not dependent on the training data available.

Since glass and other transparent objects can not be detected by LiDARs, they present considerable challenge to navigation. This dissertation presents algorithms for detecting and mapping glass objects in a Graph SLAM framework. A simple and computationally inexpensive glass detection scheme is deployed for achieving this. The efficacy of this approach via mapping of environments of different scales is showcased. Finally, the issue of *drift accumulation* that can affect mapping performance when operating in large environments is addressed by introducing a scheme that keeps an account of the locations of the glass point with respect to the sensor and mapping them by utilizing the updated trajectory estimate.

1.6 Dissertation Outline

In this dissertation, autonomous navigation, machine learning-based identification of vehicle dynamics and the indoor mapping problems are introduced in Chapter 1. In Chapter 2, a supervised learning framework is employed to learn autonomous navigation capability via deep learning. The network is trained on data generated in simulations [53] by using an off-the-shelf MPC [19] for car-like robots as the *expert* or label generator. Next in Chapter 3, the universal differential equations framework is leveraged for learning unknown vehicle dynamics efficiently and accurately. The learnt network is used for prediction the system evolution and then deployed in linear MPC based controller to showcase its efficacy.

In Chapter 4, a glass detection and mapping technique using a 2D LiDAR in a Graph SLAM framework is presented. A simple and computationally inexpensive glass detection scheme for detecting glass objects is utilized and the methodology to incorporate the identified objects into the occupancy grid maintained by the SLAM algorithm is presented. In Chapter 5, a framework for dealing with the errors introduced due to *drift accumulation* is presented.

Chapter 2: Deep Learning based Autonomous Navigation in Dynamic Environments

2.1 Overview

In this chapter, an optimization based planner, namely TEB local planner, as the expert to generate high quality motion commands for a car-like robot operating in a simulated dynamic environment. These labels are then used to train a deep neural network that learns to navigate. The deep learning based planner is further augmented with safety margins to enhance its effectiveness in collision avoidance. Finally, the performance of the proposed method is experimentally evaluated by comparing it with deployable TEB planner based local planners in the $stage_ros$ simulator as well as real world experiments. The proposed method demonstrated superior performance in terms of obstacle avoidance as well as successful mission completion. This work was published as [54].

2.2 Introduction

With the proliferation of robotics technology over the recent decades, robots have become ubiquitous in the modern world. From service robots cleaning floors of houses to UAVs performing remote surveillance, robots are touching human lives in numerous ways. As this

interaction with the world increases, there is a greater need for solutions that can enable robots to navigate safely, reliably and efficiently in these environments. While navigation in static environments can be performed satisfactorily, for the most part, several robot tasks occur in environments that are not static. Such an environment which has a fixed map, but has moving agents, will be referred to as a dynamic environment. A number of factors make the problem of navigating in dynamic environments challenging. The decisions made by other agents may only be partially observable to the robot. The robot may have conflicting objectives, imperfect or no knowledge of other agents' goals as well as planned routes [27] [31].

Autonomous mobile robot navigation has typically been dealt in a hierarchical fashion. A higher level global planner is responsible for generating a coarse plan to the goal. Local sub-goals derived from the global plan or a segment of the global plan are then fed to the local planner. The local planner generates motion commands that are kinodynamically feasible, avoid obstacles and achieve specified motion criterion [1]. For obstacle avoidance in dynamic environments, methods based on velocity obstacles [15] [55], Dynamic Window Approach (DWA) [11] and artificial potential fields [56] among others have been proposed. Recently, several data driven approaches, namely supervised learning and reinforcement learning, have been utilized for autonomous navigation in dynamic environments. Some of these techniques provide an end-to-end solution to the autonomous navigation problem [27][57] while others tend to follow the classical hierarchical paradigm [35][58] where one of the subsystems (often the local planner) utilizes machine learning. A majority of the aforementioned learning based techniques are developed using deep reinforcement learning and consequently require extensive simulations for training.

However, most of the existing works address autonomous navigation for differential drive

robots. For car-like robots, the control space is further restricted by the constraint that the ratio of the translational speed and the angular velocity is always greater than a minimum value (the minimum turning radius), which itself depends on the geometry of the robot and its steering angle. An optimization based approach aimed at enhancing comfort for objects or passengers carried by car-like robots is proposed in [59]. In [60], a feedback control law that satisfies constraints arising from obstacles, steering angle and delivers continuous control inputs is presented. Both of these approaches have been shown to work in static environments. A deep neural network based algorithm (Dynamic MPNet) that plans paths between the current position to intermediate goals from the global planner while adhering to the kinodynamic constraints of a Dubin's car model has been proposed in [61]. The authors demonstrate in [61] that the performance of Dynamic MPNet is superior when compared to DWA and Anytime RRT* [62]. However, this method too can only address navigation in static environments.

A Time Elastic Band (TEB) based approach that minimizes the time to goal while obeying the kinodynamic constraints of the robot as well as avoiding obstacles has been proposed in [19]. This approach is real-time deployable and can be used for reliable navigation of car-like robots in static environments [63]. Considering dynamic obstacle avoidance, a hybrid approach using Time Elastic Bands and artificial potential fields is presented in [64] for car-like robots. Simulations are conducted to showcase the effectiveness of the approach in a dynamic scenario, albeit a simple scenario that has a single moving obstacle. Recently, TEB planner has been extended for navigation in dynamic environments, provided accurate position and velocity estimates of dynamic obstacles are available [65]. However, the estimation of obstacle position and velocity trajectory is a challenging task. A novel method of generating a Dynamic Obstacle Layer to incorporate dynamic obstacles as virtual static obstacles has been proposed in [66].

However, it requires accurate velocity information of the objects, which is a hindrance in real-world applications. It should be noted that the online optimization problem places a considerable computational overhead on the often resource constrained mobile robot platforms [66, 67].

In this work, a learning based local planner with a safety module for car-like robots that can reliably navigate in the presence of dynamic obstacles is investigated. The proposed method utilizes a trained neural network at run time for generating motion commands which leads to a constant query time that is independent of the scenario. A deep neural network, named CARDYNET, which utilizes convolutional neural networks (CNNs) [68] with residual connections as well as Long-Short Term Memory (LSTM) [69] layers is proposed. The model is trained on data collected via simulations in $stage\_ros$ simulator [53] where the TEB local planner is provided with ground truth data for dynamic obstacles to generate high quality motion commands.The choice for the TEB planner over the DWA planner was based mainly on the fact that the latter is suitable for differential drive and omnidirectional robots only, while the TEB planner can handle car-like robots too [12]. Once trained, the performance of the deep neural network is compared with the TEB planner augmented with an open source available technique for handling navigation among dynamic obstacles [65].

The main contributions of this work are summarized as follows. A deep learning based local planner for car-like robots to handle dynamic environments is developed. The learned model leverages high quality labels generated by an optimization based framework that takes the kinodynamic constraints of the platform into account.Note that the TEB planner is used for generating the expert labels and the objective is to mimic the immediate control action to be taken rather than the entire sequence. The addition of carefully constructed LSTM-based layers to the

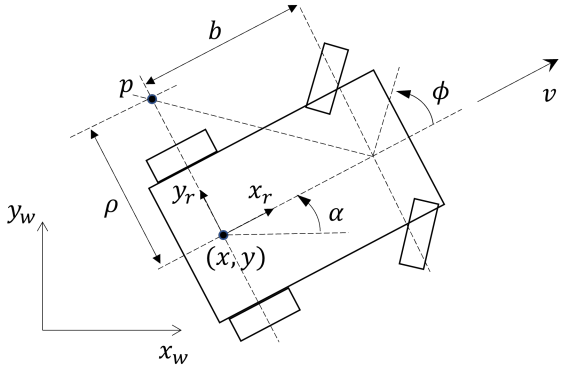


Figure 2.1: Car-like robot model.

deep learning model enables handling of dynamic obstacles without requiring a separate motion tracking module. A reactive safety module that works in synergy with the machine learning based planner is utilized. Using extensive experimentation in heterogeneous environments, the efficacy of both the end-to-end machine learning (DL planner) and safety enhanced machine learning planner (DL+safety planner) is demonstrated.

The rest of the chapter is organized as follows. The car-like model, the supervised learning framework for regression, and the TEB local planner are introduced in 2.3. Next, the proposed deep learning based approach is described in 2.4. The experimental results are then presented in 2.5. Finally, the chapter is concluded in 2.6 by summarizing the work and discussing possible future directions.

2.3 Background

2.3.1 The car-like robot model

The Ackermann steering model will be used for modelling the kinematics of the car-like robot [70] as shown in Fig.2.1. The pose of the robot consists of the position and orientation of the robot, namely x and y coordinates of the centre of the rear axle and the orientation α (or

heading direction) with respect to the world frame (x_w, y_w). The steering angle is denoted by $\phi \in [-\phi_{max}, \phi_{max}]$ with $\phi_{max} < \frac{\pi}{2}$. The forward speed is denoted by v. The wheelbase, which is the distance between the front and back axles is denoted by b. Assuming there exists a lower-level controller to change the steering angle instantaneously, the simplified kinematics of the car-like robot are given by the following set of equations:

$$\begin{aligned} \dot{x} &= v\cos\alpha \\ \dot{y} &= v\sin\alpha \\ \dot{\alpha} &= \frac{v}{b}\tan\phi. \end{aligned} \tag{2.1}$$

The control input to this model is $u = [v, \omega]$, where $\omega \triangleq \dot{\alpha}$ is the rotational speed of the robot. These can be transformed to the command actions (v, ϕ), where v is the translational speed and ϕ is the steering angle obtained by $\phi = \tan^{-1}(\omega b/v)$.

The instantaneous centre of rotation of the vehicle is denoted by p (Fig.2.1) and the turning radius is denoted by ρ. As the steering angle is limited by ϕ_{max}, the minimum turning radius is $\rho_{min} = b/\tan(\phi_{max})$. This should be encoded as a constraint in the motion planning problem. The car-like robot cannot turn in place; that is, its orientation cannot change without the robot moving along its translational axis. This should also be captured as a constraint. When moving on a straight path or a circular segment, consecutive poses of the robot must lie on a common arc of constant curvature [19]. Finally, depending on the specifications of the robot, there are constraints on its linear and angular velocities as well as accelerations.

2.3.2 Supervised learning for regression problems

This is a class of machine learning algorithms that work with labeled data i.e. there is a dataset $D = \{x, y\}_{i=1}{}^{n}$ with n input-output pairs. If the predicted variable, y, is categorical, the problem setting is known as *classification*. Instead, if y denotes an estimate of a continuous real valued quantity, it is known as a *regression* problem[20].

Fitting a model to the dataset requires choosing a suitable *loss function*. Depending on whether it is a classification or a regression problem, the choice of loss functions maybe different. The mean squared loss (MSE) is a popular choice for regression problems. Once the loss function is chosen, the difference between the predictions of the model on inputs x and the actual labels y is expressed as an averaged sum over all the examples in the dataset. This is sum is known as the *cost function*. The aim of the supervised learning problem is to determine model parameters Θ such that the cost function is minimized. The most common approach to do so is to use some variation of the gradient descent algorithm [71]. Once trained, the model is used to make predictions on new data.

2.3.3 The TEB local planner

The Timed Elastic Band (TEB) local planner solves the motion planning and obstacle avoidance problem for the car-like robot by solving an online optimization problem. The problem is formulated as a model predictive control problem that minimizes the time interval between consecutive poses. The kinodynamic constraints are enforced as the constraints of the optimization problem. In order to render this problem tractable in real-time, the constrained optimization problem is converted to an unconstrained one. This is accomplished by

incorporating the constraints as squared penalty terms. The solution to the optimization problem consists of a sequence of poses and the time intervals between these poses. The controls can be inferred by using the kinematics of the car-like robot. The unconstrained optimization problem that the planner solves is given as [19]:

$$\mathcal{B}^* = \arg \min_{\mathcal{B} \setminus \{s_1, s_n\}} \tilde{V}(\mathcal{B}), \tag{2.2}$$

$$\tilde{V}(\mathcal{B}) = \sum_{k=1}^{n-1} [\Delta T_k^2 + \phi(h_k, \sigma_h) + \chi(\tilde{r}_k, \sigma_r) + \chi(\nu_k, \sigma_\nu) + \\ + \chi(o_k, \sigma_o) + \chi(\alpha_k, \sigma_\alpha)] + \chi(\alpha_n, \sigma_n). \tag{2.3}$$

Here $\mathcal{B}^*$ is the optimal solution vector, s_1 is the current robot pose, and s_n is the target robot pose to be attained in a sequence of n steps. Additionally, ΔT_k is the time required to transit between two consecutive poses, h_k is an equality constraint and $\tilde{r}_k, \nu_k, o_k, \alpha_k$ are the inequality constraints in the TEB optimization problem. These constraints are transformed to quadratic penalties using the functions $\phi(\cdot, \cdot)$ and $\chi(\cdot, \cdot)$ with the corresponding weights denoted by σ_i with the corresponding subscripts. This problem is solved using the $g2O$ optimization framework [18] which is an open source nonlinear-optimization solver. Refer to [19] for details of functions $\phi(\cdot, \cdot)$ and $\chi(\cdot, \cdot)$.

The TEB local planner is available as an open source package (*teb_local_planner*) on ROS (Robot Operating System) [72]. In deployment, the TEB local planner fetches local goals from a global planner which is done by truncating the global plan at a predefined 'look out' distance. A major assumption in the basic TEB local planner (base TEB) for car-like robots [19] is that the environment is static; that is, all the obstacles are stationary in the map. However, if the

local motion plan is replanned frequently, it may be sufficient to avoid obstacles moving at low velocities, yet anticipatory obstacle avoidance might not be possible [73]. Recently, the TEB local planner has been extended to support navigation in dynamic environments [65]. In principle, this is done by providing a list of dynamic obstacles and their respective velocities to the optimization solver. Although providing the accurate information about the dynamic obstacles is hard in real world deployment, the *teb_local_planner* gives the users an option to include dynamic obstacles by utilizing the history of costmaps and detecting the dynamic obstacles, and inferring their velocities via a separate *costmap_converter* package [65].

2.4 Deep learning

Here, the replacement of the expensive optimization based local path planner in the navigation stack by a deep neural network is proposed. Deep learning models over the past decade have been used to deliver data driven solutions to various challenging problems [20]. In particular, the CNNs have proved particularly useful for computer vision tasks such as object detection and image classification. Another class of deep learning models called the LSTM [69] have resulted in a powerful tool for modelling temporal dependencies in applications such as machine translation, text summarization and time-series prediction.

2.4.1 Problem formulation

The objective is to find a function $u = \mathcal{F}_\Theta(L, g)$ that maps the inputs L and g, which are the sensory data and the local relative goal data, respectively, to control inputs u assuming such a relationship exists. Here the function $\mathcal{F}_\Theta(L, g)$ will be modelled as a deep neural network

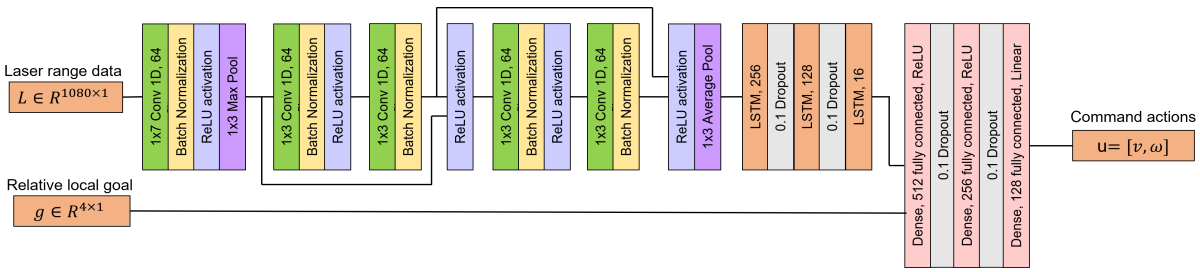


Figure 2.2: The deep neural network architecture (CARDYNET) with LSTM layers used to predict the control inputs. The inputs to the network are the laser range data and the relative local goal data. The output of the network is the control inputs.

parameterized using the parameter vector Θ. Given the training data L_t, g_t, u_t, where u_t are the ground truth commands from an expert planner, the model parameters will be optimized during training based on the mean squared error (MSE) $E = \frac{1}{n}\sum_{i=1}^{n}(F_\Theta(L_{t_i}, g_{t_i}) - u_{t_i})^2$ for the training instances n.

2.4.2 Deep learning architecture

2.4.2.1 The Network Model

A combination of CNNs with residual connections along with LSTM layers for processing the laser scan data is employed. The CNN architecture is similar to that used in [24] where the authors aim to achieve end-to-end navigation in static environments. The output from the CNN layers is connected to three LSTM layers and the output from this branch is concatenated with the relative local goal data, and is fed to a fully connected layer. The output of the network is a two dimensional vector, $u = [v, \omega]$, containing the predicted translational velocity, v and the rotational velocity, ω. The complete network architecture is illustrated in Fig. 2.2.

2.4.2.2 The inputs

The inputs to the deep learning model consist of the laser range data, $L \in \mathbb{R}^{1080\times1}$, and the relative local goal information relative to the robot (position and orientation) in the horizontal plane, $g \in \mathbb{R}^{4\times1}$. The laser range data, L, is a 1D array of 1080 values which is the range recorded at each $0.25°$ within a $270°$ field of view (FOV) with the 540^{th} element being the range corresponding to directly in front of the robot. For the purposes of this work, the scanning range was restricted to 5m to only provide the local information. The local relative goal, g, is calculated using the robot's pose and the local goal. The local goal is fetched from the global planner by trimming the global plan at a distance of 5m from the robot. Both, the robot pose and the local goal are spatially represented as 3D cartesian cordinates (x, y, z) for position and a quarternion ($\mathbf{q} = q_r + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k}$) for the orientation. Since the current work only considers 2D motion, the z-coordinate as well as the q_i and q_j coordinates in the quarternion are ignored. Therefore, the local relative goal is expressed as, $g = [\Delta x, \Delta y, \Delta q_k, \Delta q_r]^\top$, that is, the difference between considered elements in the robot pose and local goal.

2.4.3 Data generation and simulation environments

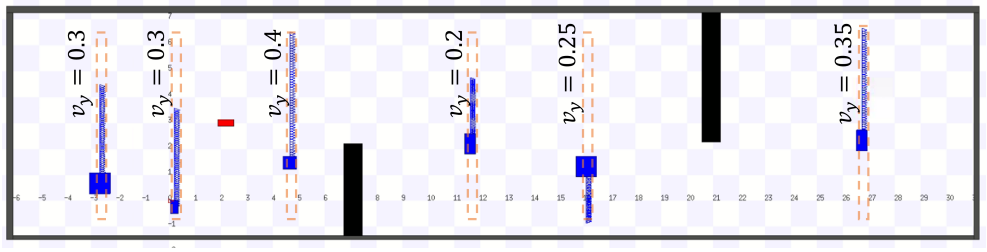
Simulations were used to generate the data for training. The environments used for the data collection were hosted in ROS [72] using the $stage\_ros$ simulator [53]. The robot chosen utilized the kinematics of a car-like robot as explained in Section 2.1. The robot parameters were selected to match those of an actual car-like robot, a commercially available MIT RACECAR robotic platform equipped with a Hokuyo UST-10LX Scanning Laser Rangefinder (LiDAR). The TEB planner was used as the expert local path planner with the dynamic obstacle avoidance

module activated. The obstacle velocities were set using a python script which enabled their precise knowledge. These velocities are utilized by the TEB planner to account for the future motions of the obstacles, and thus their accurate measurement is critical to generate safe and efficient motion plans. This also circumvented the need for a separate estimation module for tracking the obstacles and estimating their velocities. Moreover, multiple parallel trajectories were planned while running the simulation to account for the cases where the selected trajectory leads the robot to be stuck. As mentioned earlier, this is a limitation that is faced by resource constrained platforms and by utilizing the superior computing power available during offline training data generation, this shortcoming can be overcome. The global planner used in this work was A* [74] and the local goal information required for the TEB planner was fetched from the global plan.

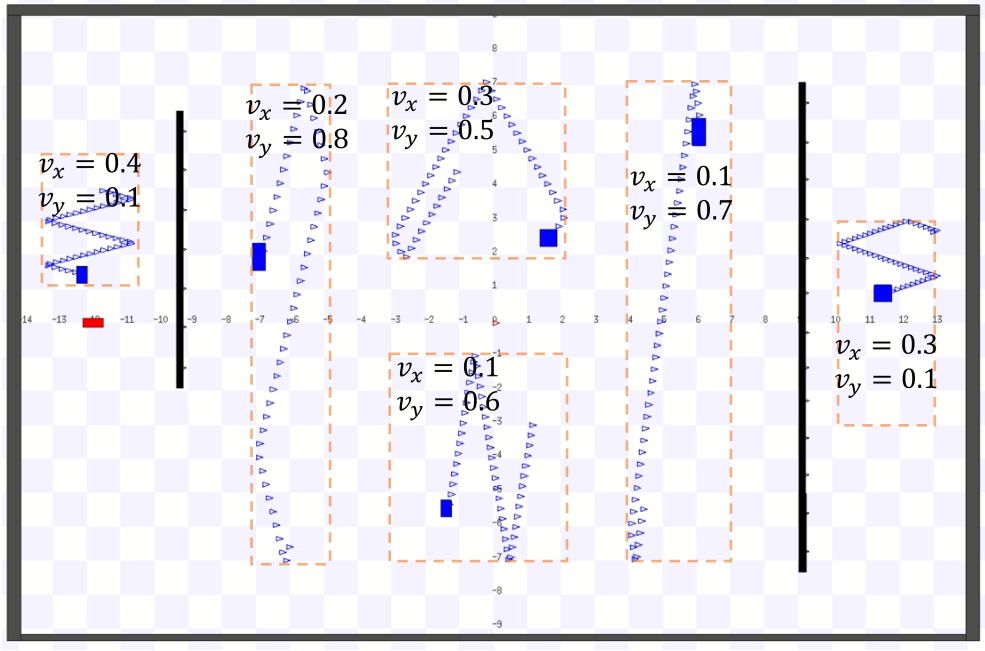
Two simulated environments were used for data collection: (i). a corridor like environment (Fig.2.3a) and (ii). a hall-like environment (Fig.2.3b). The robot is shown in red. The moving obstacles are shown in blue and the static obstacles and the boundaries are shown in black. The triangles represent the trail of the obstacles during the captured trial. The orange dashed boxes indicates the obstacles' regions of movement along with the side-ways (v_x) and up-and-down (v_y) speeds of the obstacle. The goal points were set outside the orange dashed boxes randomly. The dynamic obstacles in the corridor environment were restricted to move only up-and-down, while the obstacles in the hall-like environment were not restricted to a particular direction. However, the motion was restricted to specific pre-defined regions in order to avoid inter-obstacle collisions.

The simulations were run on Ubuntu 16.04 on an Intel® Core™ i5-3470 CPU @ 3.20GHz PC. The simulation time-step was set to 100ms, so that 10 data instances were generated per second. Each trial for data generation was run for around 20-30 mins which generated around

12000 - 18000 data instances. In each trial, new goals were set randomly every 30s - 45s intervals in guaranteed collision free regions. The initial locations of the obstacles were changed in each trial to generate an unbiased data set. The laser range data, local goal, robot pose and the control inputs were recorded at each time-step. A total of 2.6M data instances for the corridor environment and 0.6M data instances for the hall-like environment were collected. Further, in order to enlarge the data set as well as to prevent any bias of the collected data, half of the data is augmented by rotating each data instance by 90 degrees. Thus, the final data set consisted of 4.8M data instances.



(a) Corridor environment



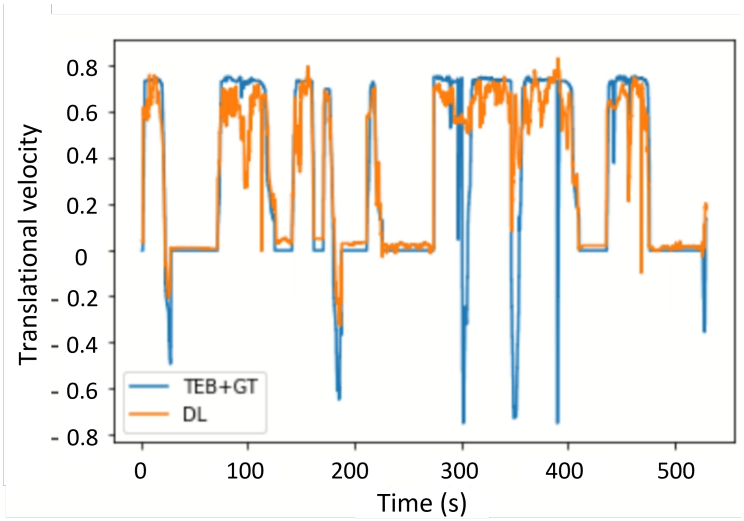
(b) Hall-like environment

Figure 2.3: Illustration of one instance of the simulated environments used for training data collection.

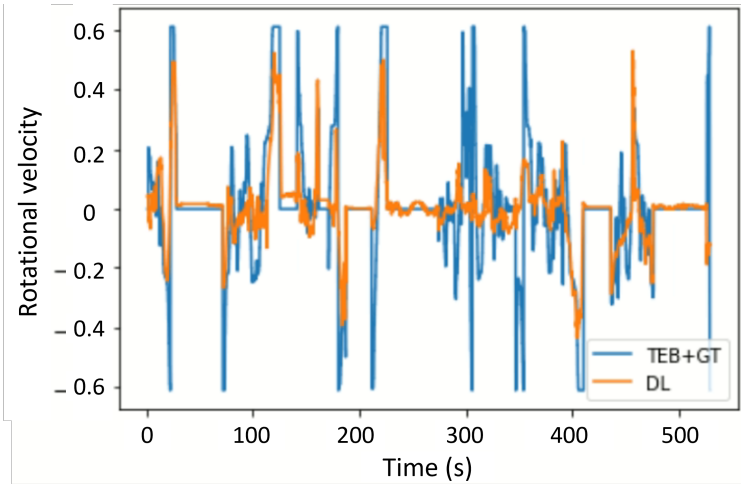
2.4.4 Model training

The whole data set was divided as 4M for training, 0.5M for validation and 0.3M for testing. The machine learning library Keras [75] was used with Tensorflow backend [76] to train the network. L2 regularization was performed on each layer to avoid overfitting.

The network is trained on a NVIDIA GeForce RTX 2070 Max-Q GPU with 8GB RAM with a mini batch size of 256. Adam optimizer was used with a loss function defined as the MSE of the mini batch. Further, early stopping was incorporated using the validation data set. The trained model, $\mathcal{F}_{\Theta^*}(L, g)$, reported a mean squared error of 0.0327 on the test set. Comparative plots of the control inputs from a subset of the test set are illustrated in Fig.2.4.



(a) Transnational velocity



(b) Rotational velocity

Figure 2.4: Comparative plots of the control inputs from a subset of the test set. The expert planner (TEB+GT) is shown in blue and the deep learning model output is show in orange.

2.4.5 Deployment

During deployment, the trained model, $\mathcal{F}_{\Theta^*}(L, g)$, with the optimum parameters, Θ^*, will be queried at each time step to predict the control inputs $u = [v, \omega]$. Additionally, a safety margin for obstacle avoidance will be used when deploying the trained model.

Safety margins: The robot will detect if an object is close by checking the laser range data. The margins will be defined in three regions, front ($L_i \leq R_f$, for any $360 \leq i \leq 720$), right

($L_i \leq R_r$, for any $1 \leq i < 360$) and left ($L_i \leq R_l$, for any $720 < i \leq 1080$) where R_f, R_r, R_l are the margins. If any object enters the safety margins, the robot will execute evasive action u_s. If an object enters the front safety margin, the robot will back-up, that is, $u_{sf} = [v_f, 0]$ until the obstacle is cleared. If an object enters the left side margin the robot will make an evasive action toward the right side moving forward, $u_{sr} = [v_r, \omega_r]$. Similarly, if an object enters the right side margin the robot will make an evasive action toward the left side moving forward, $u_{sl} = [v_l, \omega_l]$. Here the values $v_f, v_r, v_l, \omega_r, \omega_l$ need to be tuned heuristically.

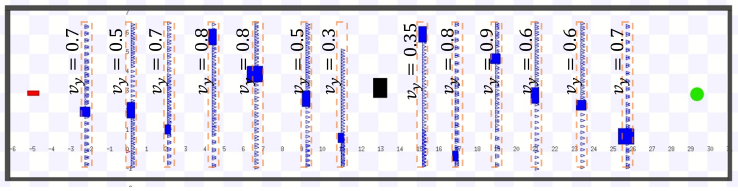
Therefore, with the safety module, the final control inputs will be $u_{cmd} = u_{sf} + u_{sr} + u_{sl} + u$, wherein the evasive actions kicked in only if an object is detected within the safety margins. If no safety module is used, the control inputs will be $u_{cmd} = u$.

2.5 Evaluation

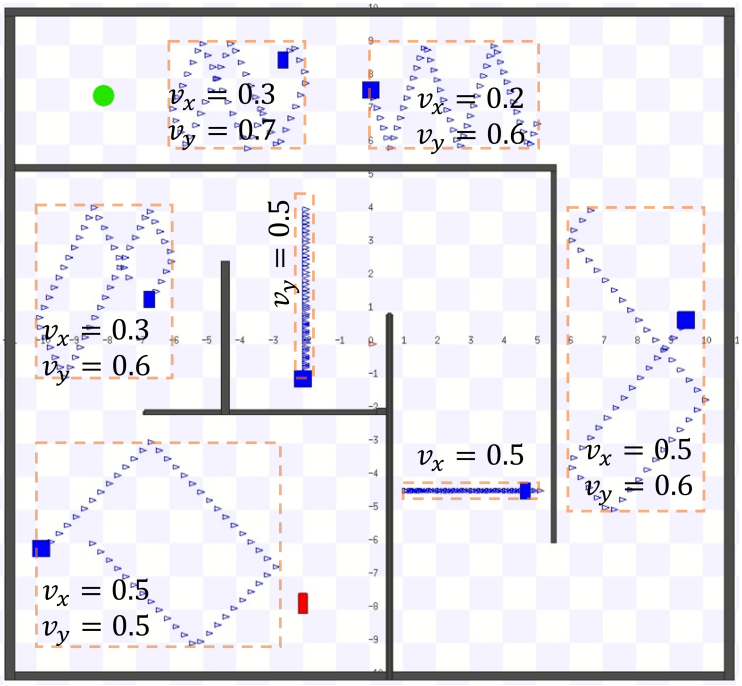
In this section, the experiments performed in order to assess the efficacy of the proposed solution in comparison with existing methods are discussed. The evaluation was conducted in both simulations and real world experiments.

2.5.1 Simulation Environments

Two distinct environments that were unseen during training, were created, to deploy and test the proposed solution in the Stage simulator. The robot parameters were kept the same as those used during training.



(a) A Busy Corridor



(b) A Maze environment

Figure 2.5: Illustration of one instance of the environments used for experimental evaluation. The robot is shown in red. The goal regions are shown in green. The moving obstacles are shown in blue and the static obstacles are shown in black. The triangles represent the trail of the obstacles in the captured trial and the orange dashed boxes indicates the obstacles' regions of movement along with the side-ways (v_x) and up-and-down (v_y) speeds.

1. **Busy corridor** This environment was created to represent a corridor scenario rich with moving obstacles (Fig.2.5a). Although a corridor like environment was utilized for generating training data, the busy corridor environment presented a higher density of moving obstacles (13 up-and-down moving obstacles shown in blue). Moreover, the obstacle velocities in this environment were higher than those utilized in training, and it also included a static obstacle (shown in black).

2. **Maze** A maze-like environment was created to evaluate the performance in an unseen environment where the robot had to make a lot of turning maneuvers in order to navigate to the goal (Fig.2.5b). This environment had 7 moving objects, 5 of which had up-and-down as well as side-ways velocities, 1 with only up-and-down velocity and 1 with only side-ways velocity.

In each environment, the starting position of the robot was fixed and the goal region(s) was pre-defined. Moreover, the regions for the obstacles to move and the obstacle speeds were also fixed. The initial position of each moving obstacle were randomized in each trial, within the defined regions. This enabled the testing of different scenarios (in terms of obstacles encountered) in the same environment. Next, the different TEB based models which were compared with the proposed solution are discussed and the simulation results are reported.

2.5.2 Models

Performance of the proposed deep learning (DL) based solution with and without the safety margins was evaluated. For the DL model with safety (DL+safety), the margins were set as 0.75m for front and 0.5m for the sides. To assess their relative efficacy, the base TEB planner, TEB planner with dynamic obstacles module using the costmap converter (TEB+CC) and the TEB planner with the ground truth obstacle velocities (TEB+GT) were utilized. The TEB+GT is the best achievable performance when the ground truth of the obstacles are provided accurately, thus it is named as the ideal case. The TEB+CC is an experimental solution provided to estimate the obstacle velocities in order to utilize the TEB planner's dynamic obstacle handling module in practical applications. Base TEB is a robust solution for tackling static environments, which

provides reactive behavior when presented with moving obstacles.

2.5.3 Simulation results

Multiple trials were conducted in each of the environments using the proposed DL models as well as the TEB planner based models. In each trial, the number of collisions suffered, successful mission completion, and the time to goal (in case the goal was reached), were recorded. Collisions were divided into three categories. 1. Head-on collisions: the robot made contact with an object from the front of the robot.2. Side-on collisions: the robot made contact with an object from the left side or right side of the robot which was within the 270° field of view (FOV) and/or had prolonged contact. 3. Grazing collisions: contacts made from the back or sides of the robot which were not within the FOV and only had instantaneous contact. In the event of a head on collision, the whole trial was abandoned and resulted only in reporting the head-on collision. For any other type of collisions, the trial continued even after the collision and the total number of incidents were reported.

Table 2.1 reports the performance of the different models in the two environments. The TEB+GT is considered as the ideal case and a comparison of the performance of the DL solutions as well as other TEB planner based solutions is made.

2.5.4 Real world experiments

2.5.4.1 The robotic platform

For performing real world experiments, the commercially available RACECAR robotic platform is utilized. It is equipped with a Hokuyo UST-10LX Scanning Laser Rangefinder

Table 2.1: Performance comparison

Model	Head-on	Side-on	Grazing	Success	Avg. time
Busy corridor environment (25 trials)					
Ideal†	0	0	1	100%	69s
Base TEB	2	35	**11**	92%	72s
TEB+CC	4	23	20	76%	98s
DL	4	29	14	84%	75s
DL+safety	**1**	**14**	13	**96%**	**68s**
Maze environment (25 trials)					
Ideal†	1	0	0	84%	103s
Base TEB	12	13	**5**	52%	**97s**
TEB+CC	**1**	11	8	**96%**	116s
DL	8	10	10	72%	108s
DL+safety	**1**	**8**	11	**96%**	111s

†Ideal case is TEB+GT.

(LiDAR), an Nvidia Jetson TX2 module, an IMU and a stereo camera. It is has an Ackermann steering mechanism and its kinematics are described by the car-like model in Section **??**.

Figure 2.6: The RACECAR UGV

2.5.4.2 Navigation in a hallway with pedestrians

The simulation-to-real transfer capability of the trained CARDYNET model was evaluated by running the outputs from the model on an MIT Racecar platform. The model was run on the same laptop that was used for training the network while all the sensory information was collected and motion commands executed on the physical Racecar platform. The trained model was tested in a hallway to test its performance for autonomous navigation among pedestrians in a real-world scenario. To compare the solution against existing TEB planner-based methods, the experiments were carried in a controlled fashion to keep the testing conditions similar across all methods being evaluated. The robot had four potential interactions with a pedestrian cutting across its path and one interaction of moving side-by-side with a pedestrian. These interactions are illustrated in Fig.2.7. The pedestrians' movement was such that they were instructed to start walking when the robot reached a particular point that was explicitly marked.

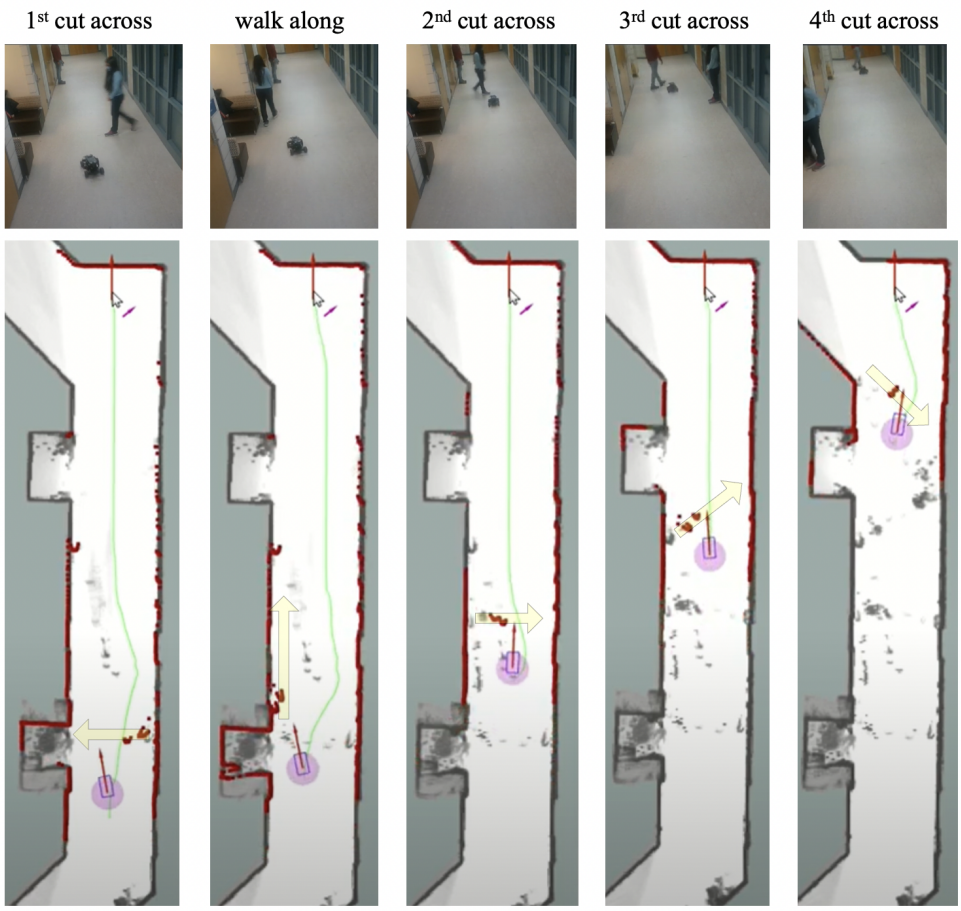


Figure 2.7: Illustration of the pedestrian interaction in the real-world experiments. The top row of images shows the camera view, and the bottom row shows the Rviz visualization of the map

2.5.5 Experimental results

Table 2.2 reports the performance of the different models over 5 trials for each model. In these experiments, the safety enhanced deep learning model (DL + Safety) outperformed its competitors. The model was the best performing as it suffered the least number of collisions as well as had the highest mission completion rate albeit compromising slightly on the average time to complete the mission. A key difference to note here is that the proposed solution only requires an initial feasible global path. Henceforth, the planner makes decisions solely based on the sensor data and the relative goal location. However, the same is not true for the TEB planner based

methods. Dynamic obstacles often leave 'trails' in occupancy grid maps which then interfere with both the local as well as the global planner. While performing the experiments, these 'trails' had to cleared by revisiting the area before a new trial could be performed. This hinders the practical applications of the costmap based TEB planner's application to navigation in dynamic scenarios.

Table 2.2: Real world performance comparison

Model	Collisions	Success Rate	Avg. time
Base TEB	2	40%	42s
Base TEB+Safety	1	40%	44s
TEB+CC	3	40%	**29s**
TEB+CC+Safety	1	80%	27s
DL	1	80%	33s
DL+safety	**0**	**100%**	39s

2.6 Discussion

In this work, an LSTM-based deep learning model with safety margins for local planning for a car-like robot is proposed. The results demonstrated that the proposed DL+safety model performs better in comparison with the other available TEB planner based methods. The experimental evaluation showed that the knowledge learned in the training environments and conditions has been satisfactorily transferred to new environments and conditions to handle dynamic obstacles. While performing real world experiments, it was observed that the DL based planner was not able to account for transparent objects. This is due to the fact that glass in not detectable in LiDAR scans except for near normal incidence. Also, the obstacles used in simulations had a constant speed. More sophisticated motion models that simulate real world movement of pedestrians can be adopted in the future.

Chapter 3: Universal differential equations based discovery of UGV dynamics

3.1 Overview

The identification of vehicle dynamics for mobile robots can be a tedious process involving first-principles modelling and measurement of several vehicle parameters. Even then, the complexity of all phenomenon at play might not be captured. With the rise of data driven methods, several machine learning based methods have been proposed for this task in recent years. A common theme among these approaches is to predict the next vehicle state given current measurements or estimates of the vehicle state, and the commanded control actions. This limits the utility of these models for predicting the vehicle behavior on the same timescale that these models are trained on. In this paper, the recent advances in the field of Scientific Machine Learning (SciML) are leveraged to learn the underlying vector field. This is achieved by learning the unknown dynamics of the vehicle by embedding a neural network in the dynamics specified by a set of ordinary differential equations. The known dynamics are modelled as usual. This formulation leads to what are known as universal differential equations (UDEs) that incorporate existing system knowledge along with the unknown parameterized by the network parameters. Along with learning the vector field, the UDE framework model is able to predict the motion of the vehicle over longer time horizons than it was trained on. The applicability in a model predictive control (MPC) setting is also showcased where the neural

network acts as the predictor for the unknown dynamics terms.

3.2 Introduction

Many of the motion planning and control systems for UGVs in use today utilize a MPC framework model that depends on a dynamics model [40]. Thus, the accuracy of the underlying dynamics model affects the MPC performance. Some methods circumvent this by shortening the time between the successive MPC updates. This usually comes at the cost of increased onboard computational requirements as well the need for more power to sustain the operation of the robot [77]. Also, the need for accurate dynamics model often arises in cases where the vehicle is operated at the extremes of its capabilities as the estimates provided by simplistic motion models may deviate significantly even over a short time horizon [40].

Machine learning is widely being adapted in mobile robotics for motion control. Recently, end-to-end deep learning based approaches have been proposed which aim to circumvent the need for separately modelling the system dynamics. Instead, they generate motion commands by ingesting propioceptive (IMU) and exteroceptive (camera, LiDAR etc.) sensor data. Such approaches tend to have a high sample complexity as they require huge amounts of high fidelity and diverse training data to work in real world scenarios. Another shortcoming that is widely mentioned is the lack the safety guarantees provided by traditional control schemes [44].

The Learning Model Predictive Controller(LMPC) proposed in [41] learns from the data recorded during each lap around the racetrack to improve an affine time-varying model. The model learning strategy in the aforementioned work is improved upon in [42] by utilizing a nominal model and learning the error dynamics with collected data. The use of Gaussian Process

(GP) models for modelling vehicle dynamics was proposed in [43]. This was further developed in [40] to build upon an extended kinematic model using the residual physics which is captured by a GP model. A more comprehensive study of GP models for this problem that takes into consideration the kernel choice, sample sample size and different racing conditions is presented in [39]. The use of ANNs for learning forward kinodynamics for high speed robot control was proposed in [44] where a 6 layered ANN with 256 units per layer was used.

The recent years have seen the rise of Scientific Machine Learning (SciML) techniques that pair state-of-the-art numerical methods with machine learning models. Historically, these two approaches have be seen as two separate means for systems modelling. The traditional scientific or mechanistic modelling relies on known physical laws, oftentimes making simplifying assumptions in order to keep computations tractable. On the other hand, modern machine learning techniques (especially deep neural networks) require large amounts of data to model complex phenomenon. The SciML approach aims to take the ”best of both worlds” approach by pairing the efficiency of mechanistic models with the approximation power of ML models. One such approach is the universal differential equations presented in [78].

In this work, the UDE framework to data driven modelling of a wheeled front steered robot is applied. In particular, the body slip angle and the angular velocity terms of the robot dynamics are modelled as these are in general, difficult to model compared to the precisely known kinematics terms. The incorporation of the known terms makes the discovery of unknown terms more efficient. Further, the generalization of the learnt model to prediction over a longer time horizon and its application in a MPC-based controller is showcased.

The main contributions of this work are summarized as follows,

1. Propose a UDE-based vehicle dynamics identification strategy that directly models the vector field

2. Provide a simple and easy to obtain computation of the learnt model's jacobian that can be incorporated into a MPC framework without the computational overheads that accompany traditional neural network modelling

The rest of the paper is organized as follows. Section 3.3 introduces the common vehicle models relevant to this work and the UDE framework. The details of the proposed UDE based vehicle dynamics identification method is discussed in Section 3.4. Section 3.5 presents the experimental results. Finally, in Section 3.6 the conclusion and possible future directions are discussed.

3.3 Background

3.3.1 Vehicle Dynamics Models

In this section, three popularly used models namely, the kinematic, the extended kinematic and the dynamic single track model are discussed. The pair of front and rear wheels are each modelled as a single wheel, respectively, with their masses lumped as a single mass. The motion is assumed to be planar, and it is assumed that the vehicle is rear wheel driven (no longitudinal force acts on the front wheel) [39].

The state of the vehicle consists of its locations, x and y with respect to an inertial frame S, longitudinal speed v, steering angle δ, its orientation ψ with respect to the inertial x-axis, its angular velocity ω, and a body slip angle β as shown in Figure 3.1. The forces acting on the

vehicle consists of a longitudinal force $F_{r,x}$ (in the body frame) that acts on the rear wheel, and lateral forces, $F_{f,y}$ and $F_{r,y}$, also in the body frame, acting respectively on the front and rear wheels. The corresponding wheel slip angles for the front and rear wheel are given by α_f and α_r, respectively. The slip angle is the angle formed between the velocity of wheel and its orientation with respect to the inertial x-axis. The control actions consist of longitudinal acceleration a_{long} and a steering velocity v_δ.

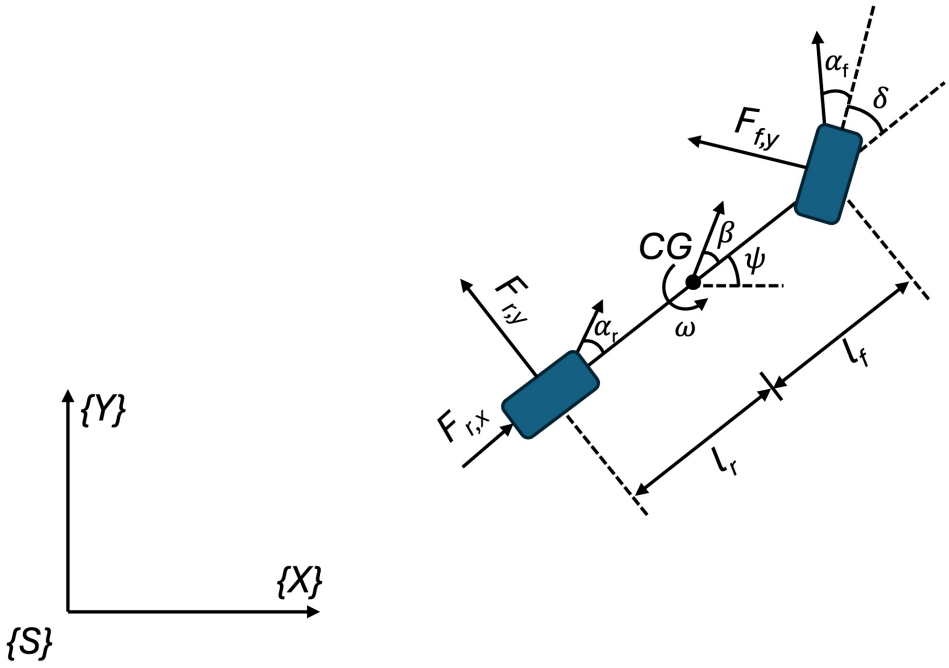


Figure 3.1: The dynamic single track model

3.3.1.1 Kinematic model

The kinematic model is the simplest among the three, as it involves the measurement of only two vehicle parameters for its specification which are relatively easy to measure. These are the distances of the front and rear axles from the center of gravity of the vehicle, denoted by l_f and l_r, respectively. This model might be preferred in some applications for its simplicity and

may work well at lower operating speeds. Mathematically, the model is represents by the set of equations 3.1.

$$\begin{aligned}
\dot{x} &= v\cos\psi \\
\dot{y} &= v\sin\psi \\
\dot{\delta} &= v_\delta \\
\dot{v} &= a_{long} \\
\dot{\phi} &= \frac{v}{l_f + l_r}\tan\delta
\end{aligned} \tag{3.1}$$

3.3.1.2 Dynamic model

While executing aggressive maneuvers or operating at high speeds, the kinematic model proves insufficient at capturing relevant effects such as oversteer and understeer [39]. The dynamic model [79], as specified by 3.2, is able to account for these effects by considering tire forces and the body slip angle. On the flip side, however, this model involves significantly more effort for tuning the various vehicle parameters [77].

$$
\begin{aligned}
\dot{x} &= v\cos\left(\psi+\beta\right)\\
\dot{y} &= v\sin\left(\psi+\beta\right)\\
\dot{\delta} &= v_{\delta}\\
\dot{v} &= a_{long}\\
\dot{\psi} &= \omega = \frac{v}{l_f+l_r}\tan\delta\\
\dot{\omega} &= \frac{1}{I_z}\left(l_f F_{fy}\cos\delta_f - l_r F_{ry}\right)\\
\dot{\beta} &= \frac{1}{mv}\left(F_{fy}+F_{ry}\right)-\omega
\end{aligned}
\tag{3.2}
$$

For the expressions for the front and rear tire forces, the reader is referred to [80].

3.3.1.3 Extended kinematic model

The extended kinematic model reduces the number of tuning parameters significantly while still accounting for the angular velocity and the body slip angle. This model is easy to specify, and requires three physically measurable parameters. In addition to l_f and l_r, the mass of the vehicle, m, is also required. When compared to the dynamic model, it ignores the effect of the lateral tire forces, $F_{f,y}$ and $F_{r,y}$. This results in a different representation of the $\dot{\omega}$ and $\dot{\beta}$ terms from that of the dynamics model involving the aforementioned forces. In the extended kinematic

model, the following representation is used,

$$\dot{\omega} = \frac{1}{l_f + l_r}\left(\dot{\delta}v + \delta\dot{v}\right)$$
$$\dot{\beta} = (l_f + l_r)\left(\dot{\delta}v + \delta\dot{v}\right) \tag{3.3}$$

3.3.2 Universal Differential Equations

Universal differential equations (UDEs) is a framework that belongs to the scientific machine learning (SciML) paradigm. It aims to leverage structural knowledge of mechanistic models along with the generality of modern machine learning methods [78]. This framework can be thought of as a variant of the more popular Neural Ordinary Differential Equations (NODEs) which rose to prominence as a result of the foundational work in [45]. It was observed that the highly effective residual connection in a ResNet architecture resembled an continuous ODE solver in the limit of infinite network depth. UDEs extend this idea by incorporating an inductive bias in the form of pre-existing knowledge of system dynamics [81]. In UDEs, only the unknown system dynamics are modelled by a feed-forward neural network. The training of the neural network parameters is carried out by minimizing a loss function that compares the trajectories of the system described using the above hybrid model and ground truth system data [81]. The training procedure is summarised in Figure 3.2.

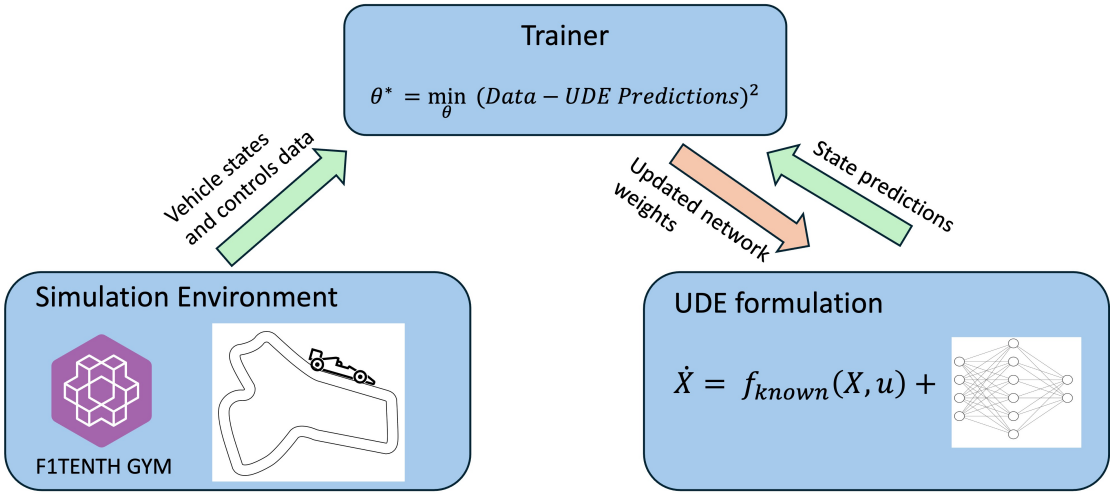


Figure 3.2: UDE-based model discovery

3.4 Methodology

3.4.1 Notation

At time t, the system state is denoted by X_t. As introduced in Section 3.3, $X_t = [x_t, y_t, \delta_t, v_t, \psi_t, \omega_t, \beta_t] \in \mathbb{R}^7$. The control action at t is given by $U_t = [v_\delta, a_{long}] \in \mathbb{R}^2$, and the output of the feed-forward neural network that models the unknown system dynamics is denoted by $\hat{Z}_t \in \mathbb{R}^2$.

3.4.2 UDE based modelling of vehicle dynamics

For the UDE formulation, a partial knowledge of the vehicle dynamics is assumed. Particularly, the dynamics of the first 5 states which are not dependent directly on the forces acting on the system are known precisely. This helps in reduction of the size of the learning problem (2 vs 7 unknown terms). The dynamics of the angular velocity and the body slip angle using a neural network. The neural network used here is a single layer feedforward neural

network with an input of length 7. The input consists of the system state (except for the 2D location of the vehicle) and the control actions at that time. The hidden layer consists of 28 neurons with hyperbolic tangent activation. The outputs of the network, Z_1 and Z_2, are embedded into the system dynamics (shown in Eqs 3.4 below). Once trained on the observed data, these hybrid dynamics can used in conjunction with a differential equations solver to accurate predict system evolution. Also, this knowledge can be directly used in MPC framework that utilizes a fourth order Runge-Kutta solver.

$$
\begin{aligned}
\dot{x} &= v\cos\left(\psi+\beta\right) \\
\dot{y} &= v\sin\left(\psi+\beta\right) \\
\dot{\delta} &= v_\delta \\
\dot{v} &= a_{long} \\
\dot{\psi} &= \omega = \frac{v}{l_f+l_r}\tan\delta \\
\dot{\omega} &= Z_1\left(\theta,\delta,v,\psi,\omega,\beta,U\right) \\
\dot{\beta} &= Z_2\left(\theta,\delta,v,\psi,\omega,\beta,U\right)
\end{aligned}
\tag{3.4}
$$

3.4.3 Loss function

The core idea behind training the network weights is to enable the UDE framework is to accurately predict the system evolution while the unknown dynamics are modelled by the aforementioned neural network. To achieve this objective, a weighted mean squared error is used as the loss function to be optimized. Precisely, if for a trajectory $\mathcal{T}$, a dataset

$\mathcal{D}_{\mathcal{T}} = \{X_k, U_k, X_{k+1}, \forall \text{ k} \in \{0, 1, ..., N\}$, then the loss function is given by

$$\mathcal{L}(\Theta) = \frac{1}{N} \sum_{k=1}^{N} w_1(\omega_{pred}^k - \omega_{true}^k)^2 + w_2(\beta_{pred}^k - \beta_{true}^k)^2 \tag{3.5}$$

where ω^k and β^k are the angular velocity and the body slip angle at time k. It should be noted that the UDE based dynamics are used to predict the next state at every time instant rather than predicting the entire sequence of N steps. It was noticed that this choice resulted in better model fitting.

3.4.4 Training data generation

The UDE is trained on data collected from the F1TENTH Gym simulator [82]. This simulator uses the dynamics model presented in Section 3.3.1.2 and the vehicle parameters from the popular F$\frac{1}{10}$ racing platform. For data collection, the vehicle is driven around a racetrack and its states and control commands are recorded every 0.01 seconds. The simulator uses a pre-computed optimal raceline computed for a given racetrack. At each instant, the state of the system accessed via the simulator and the control action is computed using a pure pursuit controller. The simulator then uses the dynamics model along with the vehicle parameters to progress to the next state. The data was collected for 2 laps around a test track and resulted in a total of 3300 datapoints.

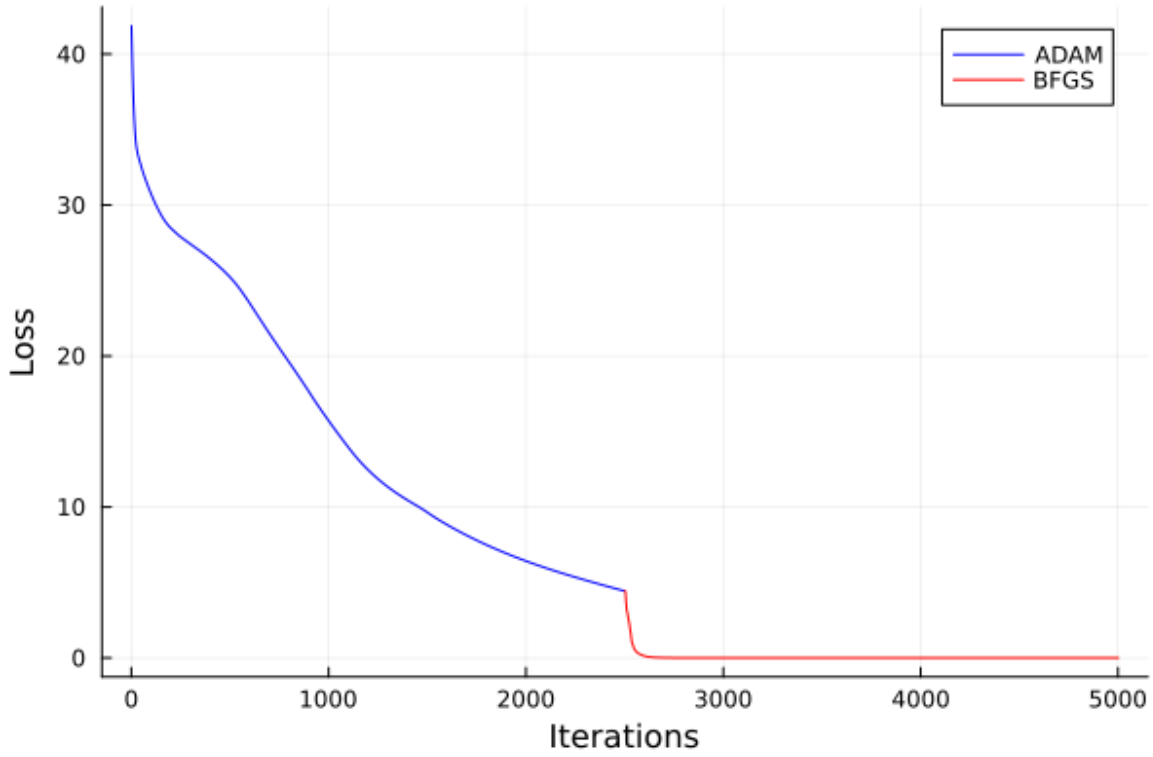


Figure 3.3: Loss function values as the training proceeds. The first half in red represents ADAM and the latter blue half pertains to BFGS.

3.4.5 Model training

The size of the training dataset for the UDE was 1650 datapoints (half of the collected data).The network parameters are trained by utilizing the loss function described in 3.5. First, the ADAM [83] optimizer is used for 2500 iterations over the data. This followed by another 2500 iterations with BFGS. The weighted MSE loss reduces from 41.86 to 2.82×10^{-5}. As described in [78], this strategy enables the search for the minima to proceed in an efficient way by first locating a good general area of search by using the *ADAM* optimizer, and finally honing in on the minimum using *BFGS*. This is shown in Figure 3.3 which shows the drop in value of the loss function with increase in number of iterations.

The trained network weights are saved and used on the remaining data (not used during

training) to asses its performance on unseen data. In this evaluation, the same loss function as used for training is utilized. The weighted MSE on the unseen data is found to be 8.18×10^{-5}.

3.4.6 Software used

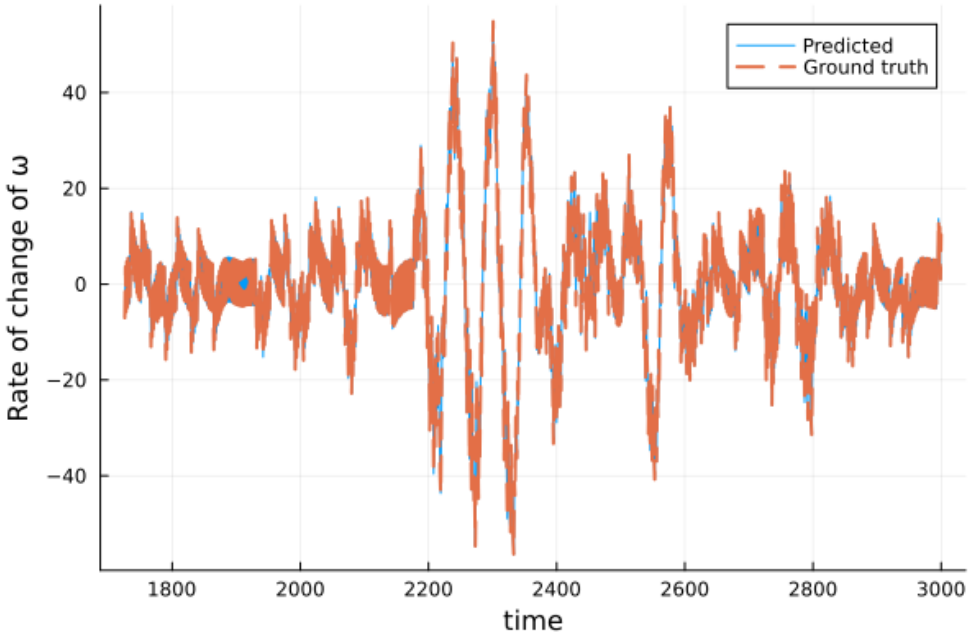
The code for this chapter was written in Julia programming language [84]. The ODE specification and solution was done using DifferentialEquations.jl [85] and the automatic differentiation and adjoint sensitivities through SciMLSensitivity.jl [78]

3.5 Results

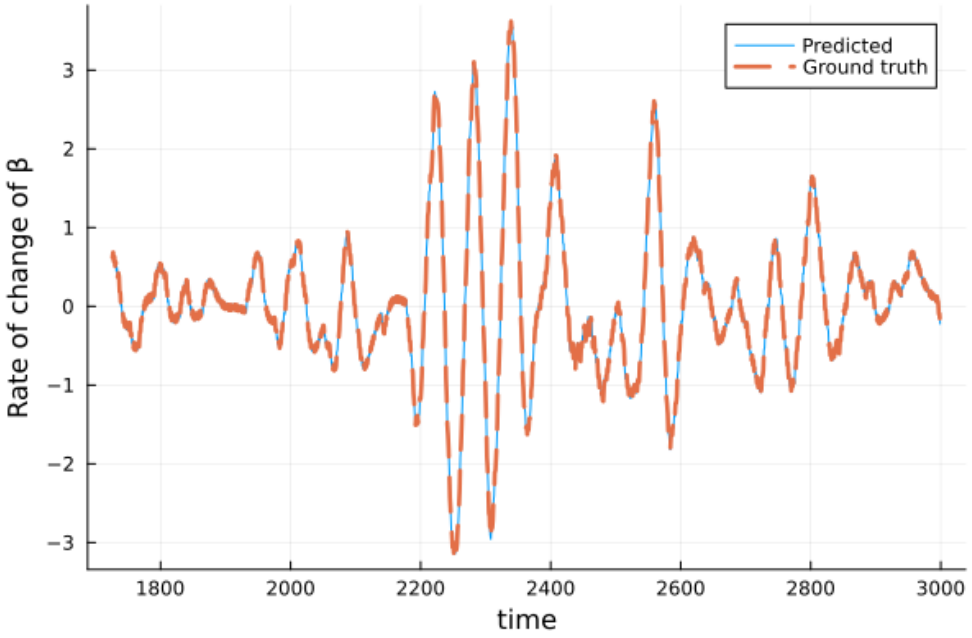
In this section, the performance of the UDE-based framework in capturing the unknown dynamics of the vehicle is assessed. Further, the trained network is used in a MPC setting and its performance is compared to the case when the MPC utilizes the true underlying vehicle dynamics.

3.5.1 Vector field modelling

A direct consequence of the manner in which the UDE is setup is that the learnt parameters Θ can be used to find the value of the vector field at any operating point. In order to assess the quality of the vector field prediction, the part of dataset that is not involved in training is utilized. The true value of the vector field is obtained by using the dynamics that underlie the simulator (this is treated as the ground truth for comparison). Figure 3.4 compares the predictions made by the trained neural network with the aforementioned ground truth values. The higher error in $\dot{\omega}$ may be attributed to its greater relative magnitude when compared to $\dot{\beta}$.



(a) Angular Velocity



(b) Slip Angle

Figure 3.4: A comparison of the predicted vector field to the ground truth values from the simulator

3.5.2 Trajectory prediction

The utility of learning the vector field compared to the next step prediction models that happens in a lot of the contemporary work is showcased by the high degree of accuracy with which the system evolution can be predicted on different horizon lengths. Even though the in the training process, the predictions of the ODE solver are compared to the ground truth over a single time step, the learnt parameters can be used repeatedly to produce the predicted value of the vector field as the system evolves. This can also be used as a dead reckoning tool in events of sensor degradation or drastic changes in environmental conditions. Figure 3.5 compares the system evolution predicted using the learned dynamics versus the ground truth over 40 time-steps.

3.5.3 Closed-loop performance in MPC

The learnt model was utilized for the predictive control task of racing the vehicle around a given track. This consists of tracking a reference optimal raceline that is pre-computed. An open source linear MPC implementation for comparing the closed loop performance of the learnt dynamics model with the ground truth dynamics is utilized. On an implementation level, the difference resides primarily in using the network output as well as its jacobian with respect to its inputs i.e. the states and control inputs. It turns out that the jacobian can be analytically be obtained and therefore does not require any additional software dependencies other than matrix multiplication and the non-linear hyperbolic tangent computation.

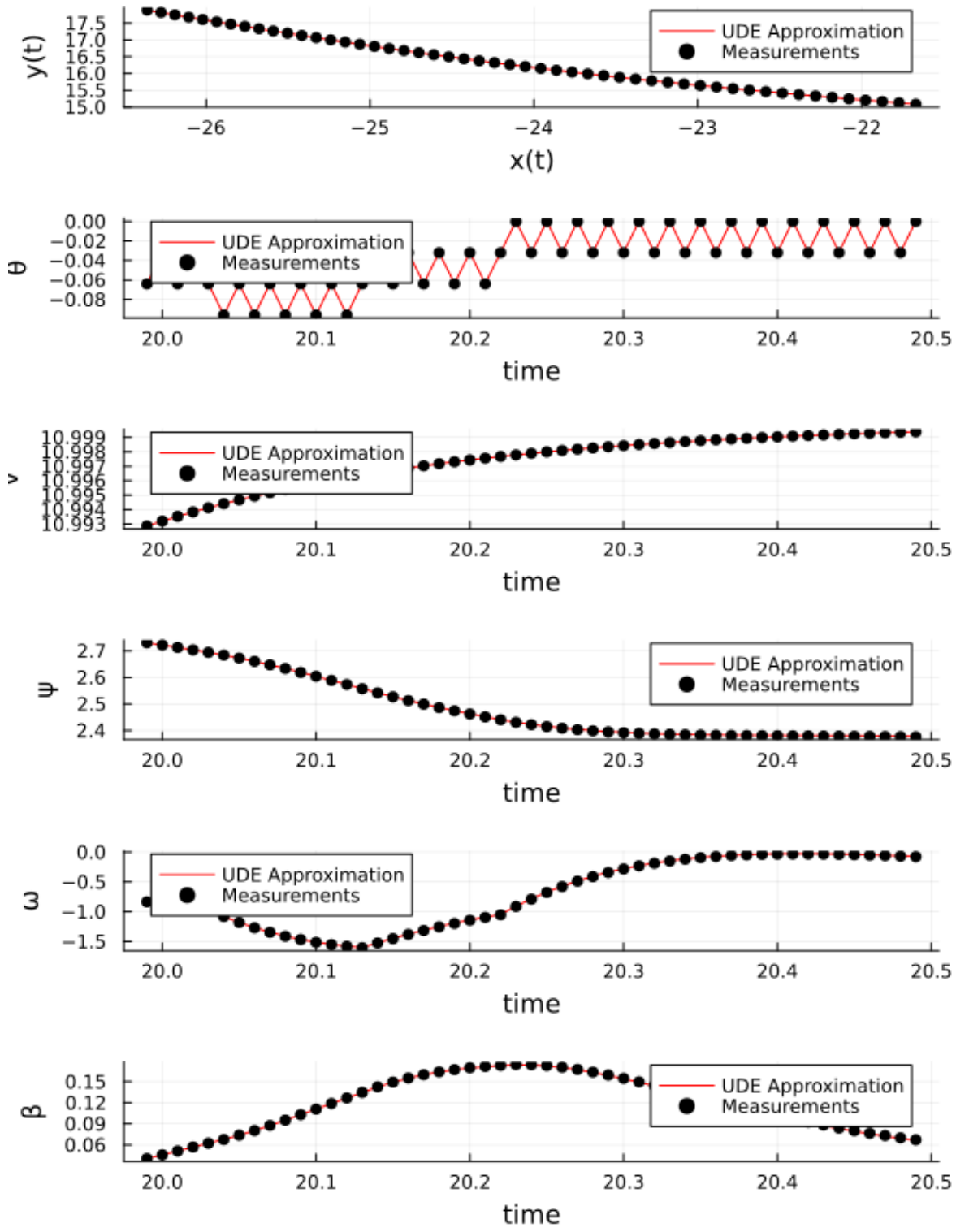


Figure 3.5: Predicting the system evolution over 40 timesteps using the learnt model and comparison with ground truth

3.5.3.1 MPC formulation

The task of racing the vehicle around a given track is formulated as a MPC problem as follows,

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & (x_N - x_{ref,N})^T Q_N (x_N - x_{ref,N}) \\ & + \sum_{k=0}^{N-1} (x_k - x_r)^T Q (x_k - x_r) + u_k^T R u_k \\ \text{subject to} \quad & x_0 = x(0) \\ & x_{k+1} = A_k x_k + B_k u_k \forall k \in \{0, 1, ..., N-1\} \\ & x \in \mathcal{X} \\ & u_{min} \leq u_k \leq u_{max} \end{aligned} \tag{3.6}$$

This problem is solved in a receding horizon fashion. Here, $\mathbf{x} = \{x_0, x_1, ..., x_N\}$ and $\mathbf{u} = \{u_0, u_1, ..., u_N\}$ are the sequence of states and control inputs, respectively. The matrix Q encodes the importance of following the specified trajectory (Q_N specifies the importance of reaching the end state). Similarly, R penalizes the use of utilizing the largest available control action. $\mathcal{X}$ specifies the set of acceptable location of the vehicle and u_{min} and u_{max} define the lower and upper bounds on control actions.

3.5.3.2 Jacobian of the neural network outputs

As described above, the jacobian of the learnt model is required for the prediction step of the MPC formulation. The network weights between the input and hidden layer at a node j of the hidden layer are denoted by $w_{j,i}^{[1]}$ where i denotes the index of the input associated with this weight and j denotes the index of the neuron in the hidden layer. b_j^1 denotes the bias term

associated with this node. Similarly, $w^2_{k,j}$ and b^2_j denote the weight and bias associated with node k of the output layer. For the network used in this work, $i \in \{1, 2, ..., 7\}$, $j \in \{1, 2, ..., 28\}$ and $k \in \{1, 2\}$. z^1_j and z^2_k denote the inputs to nodes j and k in the hidden and output layers. The activation of unit j in the hidden layer is denoted by a^1_j and is the hyperbolic tangent function applied to z^1_j while there is no activation function used at the output layer, the output a^2_k is simply equal to z^2_k for each node k.

$$
\begin{aligned}
z_j^{[1]} &= \sum_{i=1}^{7} w_{j,i} x_i + b_j^{[1]} \\
a_j^{[1]} &= \tanh z_j^{[1]} \\
z_k^{[2]} &= \sum_{j=1}^{28} w_{k,j} a_j^{[1]} + b_k^{[2]} \\
a_k^{[2]} &= \tanh z_k^{[2]}
\end{aligned}
\tag{3.7}
$$

The above quantities 3.7 are computed in the forward pass through the network and are stored for efficiency as these come handy in the jacobian computation. Since the network has 2 outputs and 7 inputs, the jacobian is 2×7 matrix. The steps involved in computing an element

$\frac{\partial a_k^{[2]}}{\partial x_i}$ are as follows,

$$\begin{aligned}
\frac{\partial a_k^{[2]}}{\partial x_i} &= \frac{\partial}{\partial x_i}\left(\sum_{j=1}^{28}(w_{k,j}a_j^{[1]}) + b_k^{[2]}\right) \\
\frac{\partial a_k^{[2]}}{\partial x_i} &= \sum_{j=1}^{28}\left(w_{k,j}\frac{\partial a_j^{[1]}}{\partial x_i}\right) \\
\frac{\partial a_j^{[1]}}{\partial x_i} &= \frac{\partial}{\partial x_i}\left(\tanh\left(\sum_{i=1}^{7}(w_{j,i}x_i) + b_j^{[1]}\right)\right) \\
\frac{\partial a_j^{[1]}}{\partial x_i} &= (1 - \tanh^2\left(\sum_{i=1}^{7}(w_{j,i}x_i)\right))w_{j,m}
\end{aligned} \tag{3.8}$$

It is important to note that this computation of the jacobian in 3.8 can be efficiently carried out by vectorizing the calculations. The implementation used in the MPC was accomplished without using any deep learning libraries, and instead relied on NumPy based matrix operations along with the hyperbolic tangent function.

The vehicle is driven around a test track for two loops and horizon windows of length 20 and 25 steps are chosen. The starting point for both sets of experiments is kept the same. Table 3.1 compares the performance of the learnt model against the implementation that utilizes the true simulator dynamics. It can be seen that the learnt model delivers competitive performance in both cases.

Model used	N = 20	N = 25
Ground truth	30.08s	28.34s
UDE-based (Proposed Approach)	30.31s	31.54s

Table 3.1: A comparison of time taken by the MPC using the ground truth versus the proposed approach

3.6 Conclusion and future work

In this work, the use of universal differential equations for vehicle dynamics identification was explored. It was shown that the dynamics can be learnt accurately and the applications for trajectory prediction and closed-loop MPC implementation were showcased. However, the set of network weights that were learnt are constant. This might not always be true and the system properties might change. Future work will entail experiments on physical robots and addressing time-varying system dynamics.

Chapter 4: Glass Detection and Mapping using a 2D LiDAR in Graph SLAM Framework

4.1 Overview

In this chapter, algorithms for detecting and including glass objects in an optimization-based Simultaneous Localization and Mapping (SLAM) algorithm are studied. When LiDAR data is the primary exteroceptive sensory input, glass objects are not correctly registered. This occurs as the incident light primarily passes through the glass objects or reflects away from the source, resulting in inaccurate range measurements for glass surfaces. Consequently, the localization and mapping performance is impacted, thereby rendering navigation in such environments unreliable. Optimization-based SLAM solutions, which are also referred to as Graph SLAM, are widely regarded as state of the art. In this chapter, a simple and computationally inexpensive glass detection scheme for detecting glass objects is utilized and the methodology to incorporate the identified objects into the occupancy grid maintained by such an algorithm (Google Cartographer) is presented. In this work, both local (submap level) and global algorithms for achieving the objective mentioned above and compare the maps produced by the proposed method with those produced by an existing algorithm that utilizes particle filter based SLAM. This work has been made available via [86].

4.2 Introduction

The development of Simultaneous Localization and Mapping (SLAM) algorithms has enabled precise mapping of indoor and outdoor environments [87]. Improvements in sensing modalities such as LiDARs and cameras now enable high-quality 2D and 3D reconstruction of environments. LiDARs, in particular, have become increasingly popular due to their precision and long-range operation. Despite these attractive features, LiDARs perform poorly in the presence of transparent objects and reflective surfaces. In modern building design, transparent walls and partitions are being increasingly used for aesthetic appeal and energy-efficient interior lighting. Therefore, detecting and including such transparent objects in the map is a must for reliable and safe indoor navigation.

A LiDAR measures the range to an object based of the reflection of the laser beam incident on it. Diffuse surfaces disperse the beam equally in all directions. As a result, the corresponding intensity of reflection from such surfaces is usually low [88]. However, when the beam is incident on a transparent object like glass, most of it passes through, and the rest is reflected away at an angle equal to the angle of incidence. This leads to the LiDAR not being able to detect the object from most angles. This not the case when the incidence angle becomes near normal. Most of the incident beam is reflected directly back to the source, and a sharp increase in intensity is observed. This intensity value is significantly higher than intensity values for reflection from diffused surfaces.

There have been several approaches in the literature that have tried to address the challenge of detection and mapping of glass and transparent surfaces. Foster et al. [88] consider that glass behaves like a diffuse object in a minimal range of angles (near-normal incidence) and

thus gets treated as noise by the SLAM framework, eventually not getting registered on the map. They augment the standard occupancy grid mapping by tracking the subset of angles from which glass objects are visible and reconstructing the objects from these angles assuming there is some localization error. In [89], a simple detection scheme is utilized by Wang and Wang by examining the intensity profile of the returned laser beam around normal incidence to the glass surface. This was then incorporated into GMapping [90] (an existing particle filter based SLAM approach) naming the package as *slam_glass*.

The use of multi-echo lasers to detect glass objects has been described by Koch et al. [91, 92]. In recent work, Kim and Chung [93] utilize the reliability of LiDAR measurements to localize a robot in glass environments. In [94], Awais proposed a probabilistic approach to model the glass surfaces using laser scans for improved laser-based navigation of mobile robots. Jiang et.al. [95, 96] used neural networks to build glass confidence maps using laser range finders. However, the focus of their work was not SLAM but rather the classification of wall surfaces as glass and non glass. Recently, Jung et.al. [97] proposed a method for transparent obstacle detection by analyzing the noise generated when a laser beam meets a transparent obstacle. In [98], Meng et.al. used maps generated from glass environments by an off-the-shelf SLAM solution, and manually labeled the glass walls for accurate LiDAR-based localization in those environments. Another common practice is to utilize additional sensors and employ sensor fusion to deal with the detection [99] and inclusion of glass surfaces or objects into the map [100, 101] [102, 103, 104]. In these approaches, sonars are commonly used as the additional sensing modality.

In this work, a framework for mapping glass environments using a LiDAR in a 2D Graph SLAM framework is proposed. Specifically, the sharp spike in intensity values to detect the

presence of glass and other transparent surfaces as introduced in [89] is exploited. These detected points are incoprrporated into an existing Graph SLAM method known as the Google Cartographer [105]. Graph SLAM methods are widely considered to be the state-of-the-art methods in SLAM [106]. Specifically, Google Cartographer has generally proven to provide robust and accurate maps over other SLAM methods [107] in 2D LiDAR SLAM. Traditionally popular methods such as particle filter-based methods utilize a set of weighted hypotheses (particles) where each particle maintains the full representation of the environment being mapped [90]. Consequently, these methods can become computationally expensive when mapping large environments [105].

In the remainder of the chapter, all the transparent *glass-like objects* will be referred to as glass. The main contributions of this work are the following:

- Implementation of glass detection and mapping in an optimization-based SLAM algorithm (Google Cartographer),
- Improved accuracy over the maps generated by a GMapping based glass detection and mapping scheme,
- Extensive testing on glass environments using a mobile platform equipped with a LiDAR.

4.3 Background

4.3.1 Pose Graph Optimization for Simultaneous Localization and Mapping

Pose Graph Optimization (PGO) is a least squares approach for solving the Simultaneous Localization and Mapping (SLAM) problem. The problem is modeled as a graph where the

poses constitute the *vertices* or *nodes* of the graph, and the constraints between the poses form the *edges* of the graph. Constraints inserted between successive nodes relate their relative poses. These maybe based upon measurements from wheel encoders, IMU data, visual odometry or even scan matching between the current scan and the map. Additional constraints in the form of loop closures are inserted between non-successive poses when the robot revisits an area it has previously been to.

This is a non-convex and typically a high dimensional optimization problem that is solved by the Gauss-Newton method. The method proceeds by a succession of local linearizations. The error function is first linearized and its derivative is computed. By setting the derivative to zero, a linear system of equations is solved and this process is repeated until convergence. Since the error term e_{ij} depends only on (ξ_i and ξ_j), the Jacobian for the linearized error function is sparse. This is significant because it allows the use efficient methods such as the Sparse Cholesky decomposition to be used. More details on PGO for SLAM can be found in [108].

4.3.2 Google Cartographer

The Google Cartographer [105] is a SLAM solution that achieves real-time mapping and loop closure using LiDAR data in 2D as well as 3D environments. This work considers the 2D case. The algorithm consists of two subsystems: a local SLAM (front end) and a global SLAM (back end). The front end is responsible for constructing local a representation of the world known as a submap. Laser scans are inserted consecutively into the submap via a process called scan matching. Scan matching consists of optimizing the pose (position and orientation) of the laser scan by aligning the current scan with the submap. Even though the scan insertions are

locally consistent, over time this process accumulates error (also referred to as *drift*). Submaps are the result of iteratively performing this alignment of the scan to submap coordinate frames. Representation wise, submaps are represented as a probability grid where each grid cell is associated with a probability value. This value expresses the likelihood that the grid cell is occupied. A laser scan is inserted into this probability grid by determining the grid points' set where the laser ray ends through a process known as *ray casting*. These grid points constitute the *hits* set. Any grid cell that the ray passes through and is not in hits set is included in the *misses* set. Previously unobserved grid points are assigned p_{hit} or p_{miss} which represent the hit and miss probabilities respectively. The grid probabilities of already observed cells are updated accordingly for hits ($p = p_{\text{hit}}$) or misses ($p = p_{\text{miss}}$) as,

$$M_{\text{new}}(cell) = \text{odds}^{-1}\left(\text{odds}\left(M_{\text{old}}\left(cell\right)\right) \cdot \text{odds}\left(p\right)\right)$$

where $M_{\text{old}}(cell)$ is the prior grid probability of the considered $cell$ and the odds for grid probability p defined as $\text{odds}(p) = \frac{p}{1-p}$.

The back end's role is to check for loop closures and deliver the most consistent sequence of global poses. The error accumulated during scan insertion in submaps is minimized by running a pose optimization, where the poses of all scans and submaps are optimized. To check for loop closure, each scan is matched with already completed submaps. If a match is found, a constraint is inserted in the optimization problem that relates the relative poses. More details on the optimization problem formulation can be found in [105].

4.3.3 Glass detection

The detection of glass surfaces is performed using only the intensity data of the reflected LiDAR beams, as introduced by the authors in [89]. Since glass surfaces are highly polished, they specular reflect most of the incident laser beam. Therefore, the received rays' intensity is highest when the robot is receiving the laser beam back from a near normal incident angle. Hence, by inspecting the laser scan data's intensity profile and identifying the intensity peak, glass surfaces can be detected. For false proof detection, this algorithm uses an intensity threshold value (*thresh*), an intensity gradient (*grad*) and a profile width (*width*) as tunable parameters. The method used for glass detection is shown in Algorithm 1. Although this detection method allows only a small range of view angles, this has been a proven method of glass detection in recent literature [88, 89].

Algorithm 1: Glass Detection

Require: Point Cloud with intensities and given *thresh*, *grad* and *width*
Initialization: $isGlass$(Point Cloud) $= 0$ and int_0 = intensity of first data point
for *all p in Point Cloud* **do**
  int_p = intensity of p ;
  if $int_p \geq thresh$ & int_p-$int_{p\text{-}1} \geq grad$ **then**
    h $\leftarrow$ p;
  else
    if $int_p \geq thresh$ & int_p-$int_{p\text{-}1} \leq grad$ **then**
      if $p\text{-}h \leq width$ **then**
        r := floor($\frac{p+h}{2}$);
        $isGlass(r) \leftarrow 1$
      end
      ;
    end
  end
end
return $isGlass$(Point Cloud).

4.4 Glass mapping in 2D Graph SLAM

In this section, the methodology for incorporating detected glass points to a 2D Graph SLAM framework, the Google Cartographer, will be discussed. The main objective of the glass mapping scheme is to add the detected glass points robustly into the map. Ideally, this could be done by identifying the cell index of the detected glass point in the probability grid and increasing the grid probability of that particular cell to the maximum allowable value. However, a glass point is visible and detected as a hit only when the incident laser scan ray is normal to the glass surface. Consider the case of a glass point g that was registered as a hit point from a specific location. When the robot moves from that location and g is no longer a point of normal incidence, the laser rays pass through it or specular reflect in a direction away from the LiDAR. The point g would then be registered as a miss. Hence, even point g's grid probability is set to maximum at the time of detection, its occupancy probability will decrease every time when it is registered as a miss (this will happen for a majority of the angles).

Therefore, to incorporate the detected points in the map resolutely, the **Cartographer_glass** framework is proposed which uses two sub-schemes. 1). The *local glass mapping scheme* maps the detected glass points onto the current submap, and 2). the *global glass scheme* retains the detected glass points in the global map.

4.4.1 Local glass mapping

In this scheme, the cell index of a detected glass point in the current submap's probability grid is first identified. Next, the grid probability of the identified cell is set to the maximum value. Consequently, the detected glass point is now added to the local submap as a 'hit' with

the maximum grid probability, encoding an identification of the observed detected glass point(this will be useful in identifying subsequent occurrences of the same point). If an already observed glass point is observed again, as a 'hit' or as a 'miss', its occupancy probability will not be updated from the set maximum value. This procedure ensures that the detected and mapped glass points on the submap do not 'fade' away. Further, the global coordinates of the detected glass points with respect to the global coordinate frame are saved in a global *Glass_points* array by transforming the coordinates given in the current (local) frame using a homogeneous transformation matrix H_k. Here H_k is the homogeneous transformation from the global coordinate frame to the coordinate frame of the k^{th} submap, which is computed while running the global optimization (back-end of the Cartographer). The *local glass mapping* scheme is shown in Algorithm 2 which will return the probability grid for the k^{th} submap.

Algorithm 2: Local Glass Mapping Scheme

Require: Range data including the detected glass points with an identifier, global *Glass_points* array and the homogeneous transformation matrix H_k .
Initialization: k^{th} submap with probability grid
while ! *Finish_Submap* **do**
  for *all* $x_i \in$ *Range Data* **do**
    cell $\leftarrow$ index(x_i);
    if $x_i =$ *Detected glass point* **then**
      $M_{\text{new}}(cell) \leftarrow max_p$ (with identifier);
      Glass_points $\leftarrow$ [*Glass_points*; $H_k x_i$];
    end
    if $M_{new}(cell) != max_p$ *(with identifier)* **then**
      $M_{\text{new}}(cell) \leftarrow$ from normal calculation ;
    end
  end
end
return probability grid.

4.4.2 Global glass mapping

The *local glass mapping* scheme is only applicable at the submap level as the probability grid is re-initialized at the start of every new submap. Therefore, the previously observed glass points' identifiers in earlier submaps no longer exist in a newly initialized probability grid. Thus, if a previously observed glass point is registered as a 'miss' in the current submap, it will result in the fading away of the already added detected glass points in a previous submap according to the probability update using p_{miss}. Therefore, it is essential to add the saved points from the global *Glass_points* array to each subsequent submap to retain the detected surfaces in the global map. However, these points need to be transformed as per the latest trajectory estimates before adding them. Next, the grid probability of these added points will be set to maximum probability (with the encoded identifier). This will enable the *local glass mapping* scheme to identify the newly added points as glass points when evolving the submap. Algorithm 3 shows the *global glass mapping* scheme.

Algorithm 3: Global Glass Mapping Scheme

Require: Global *Glass_points* array and the homogeneous transformation matrix H_k .
Initialization: new k^{th} submap
for $p_i \in Glass\_points \cap span(submap)$ **do**
    cell $\leftarrow$ index($H_k^{-1} p_i$);
    $M_{\text{new}}(cell) \leftarrow max_p$ (with identifier);
end
Local Glass Mapping Scheme;
return probability grid.

However, one significant challenge for implementing the global glass scheme is estimating the global locations of the detected glass points accurately. As loop closures happen for sections of the map upon re-observing them, the current submap might span to regions that are yet to

be re-visited. In these situations, the added detected points might have a drift corresponding to the error in the robot's actual and estimated trajectories/poses. This is more evident in scenarios in which large spaces are mapped. This happens because the possibility of introducing drifts is greater in such scenarios and once introduced the drift builds up unless corrected by a loop closure. Therefore, in such circumstances, the above *global glass mapping* scheme is ineffective. However, a practical alternative to overcome this challenge is to use only the *local glass mapping* algorithm, and increase the p_{miss} value so that the decrease in occupancy probability upon non-registration is much more gradual. This will be referred as the **Cartographer_glass_lite**.

4.5 Experimental results

Experiments were conducted using a commercially available MIT RACECAR robotic platform [109] equipped with a Hokuyo UST-10LX Scanning Laser Rangefinder (LiDAR). The SLAM algorithms were run offline on a PC using the collected LiDAR data. First, the effectiveness of the proposed SLAM algorithm is demonstrated in a laboratory setting with different transparent panels outlining the calibration procedure. Next, the generated maps using the proposed glass detection and mapping scheme for the data collected from actual office buildings will be presented.

One of the motivations for glass mapping was to be able to provide the global planner with a representation that could be readily used by its costmap without any downstream modifications. Keeping this in view, no distinction was made between the identified glass objects and non glass objects. Therefore, all maps are converted and saved as an image file that encodes the occupancy data using the *map_server* ROS node by coloring each pixel correspondingly to the occupancy

state of the cells of the map. Free cells are colored white, occupied cells are colored black whereas unknown cells are gray.

4.5.1 Detection and mapping of different transparent panels

For this experiment, two clear acrylic panels, a glass panel, and a clear poly-carbonate panel were placed in a laboratory setting. The robot was then run to collect the LiDAR data.

Before running the SLAM algorithms, experiments were performed to collect data from the different transparent panels to observe their intensity profiles and select the intensity threshold value for transparent surface detection. For each of the materials, the panel was placed at a distance of 1m perpendicular to the translational axis of the robot to ensure near-normal incidence, and LiDAR data was recorded. This was repeated from a 2m distance as well. Parameter values were selected such that it would enable the detection of all material types used. Thus, an intensity threshold of $thresh = 3000$, an intensity gradient of $grad = 500$ and a profile width of $width = 10$ were selected. The materials used for data collection are shown in Table 4.1, and the intensity profiles near the normal incident angle are shown in Fig. 4.1 along with the selected parameter values.

Table 4.1: Different materials used for collecting data

Material	Thickness (in)	Height (in)	Width (in)
Glass sheet	0.125	12	18
Clear Acrylic sheet	0.093	12	24
Poly-carbonate sheet	0.5	14	24

The map generated from the default Cartographer (without glass detection) is shown in Fig.4.2 (a) in which the surfaces pertaining to the placed panels are undetected. From Fig.4.2 (b), it can be easily seen that the Cartographer_glass algorithm was able to detect and add the

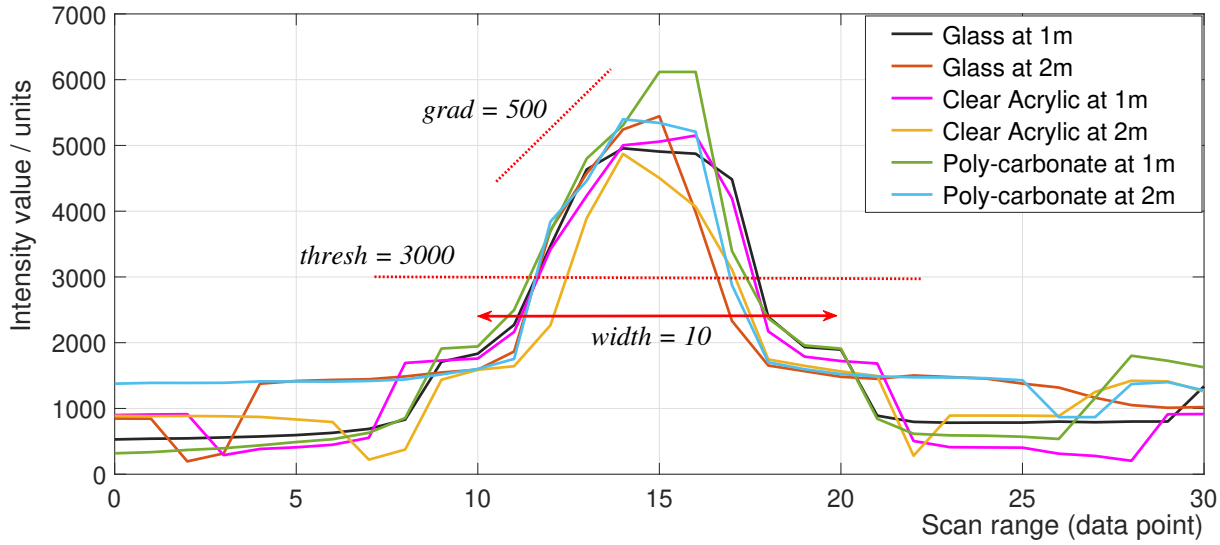


Figure 4.1: Intensity profiles from different materials and the selected parameters

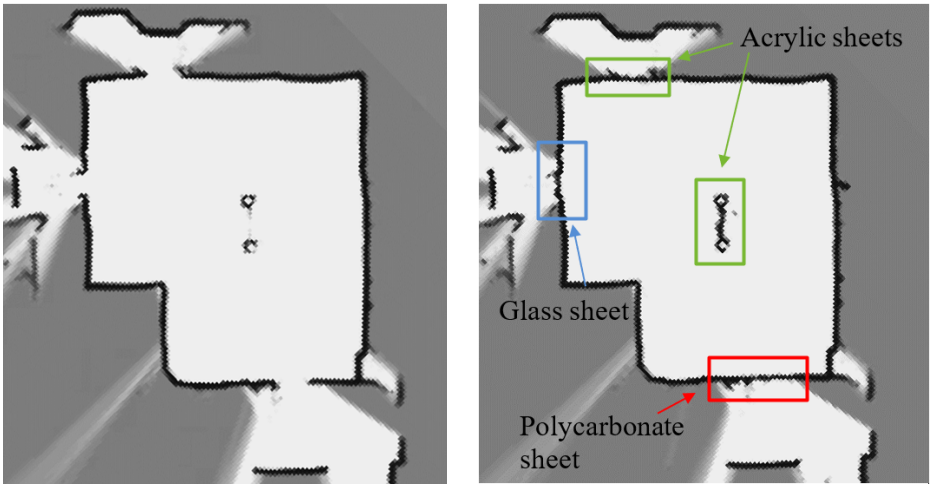
transparent surfaces to the final map.

4.5.2 Experiments in office buildings

In this section, maps generated from data collected by robots in real-world office buildings are presented. Three scenarios are considered, the data set provided by the authors in [89] and two data sets collected with the RACECAR robotic platform at the University of Maryland.

4.5.2.1 Dataset from slam_glass paper

Here, the maps generated from the data set provided in [89] are presented. The recommended parameter values of $thresh = 8000$, $grad = 4000$ and $width = 5$ are used. Fig.4.3 shows the maps generated by (a) the proposed Cartographer_glass algorithm and (b) the *slam_glass* map. By visual inspection, it can easily seen that the glass detection and mapping is more robust from the proposed Cartographer based implementation.



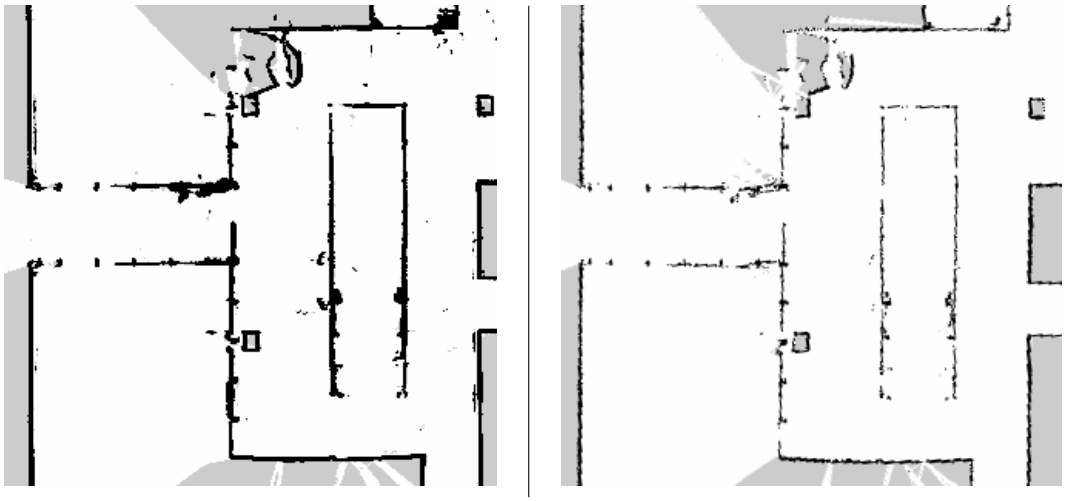
(a) Default Cartographer map (b) *Cartographer_glass* map

Figure 4.2: Maps generated (a) without and (b) with glass detection using Google Cartographer inside a laboratory setting with different transparent panels

4.5.2.2 Data collected by the RACECAR platform

The maps generated from the MIT RACECAR platform's data are presented. This data was collected from two separate buildings with several glass surfaces. Since the glass walls in these buildings were different from those used in the laboratory setting experiment, a calibration process, as discussed earlier, was repeated. The robot was placed in front of a glass wall to collect the LiDAR data, and by inspecting the intensity profile, it was determined to use the parameter values as $thresh = 1300$, $grad = 100$ and $width = 10$ for both the buildings. Furthermore, since these spaces were relatively large, the Cartographer_glass_lite framework with an increased $p_{\text{miss}} = 0.499$ was used.

First, the maps generated from the *Building 1* are presented. The ground truth for glass walls' location is manually marked in Fig.4.4 for illustration. The robot passed all the long hallways twice while the connecting short corridors were traversed twice or four times, as indicated in Fig.4.5, which is a Cartographer map generated with default settings. Fig.4.6 shows



(a) *Cartographer_glass* map (b) *slam_glass* map

Figure 4.3: Maps generated with the dataset from [89]

the maps generated by the proposed *Cartographer_glass_lite* framework and from the *slam_glass* algorithm. The parts in which the proposed Cartographer algorithm outperformed the *slam_glass* map have been highlighted and vice-versa.

Next, the maps generated from the MIT RACECAR platform's data from *Building 2* are presented. Fig.4.7 illustrates the Cartographer map overlayed with the actual floor plan of *Building 2* with the locations of glass manually marked for illustration. Fig.4.8 shows the trajectory of the robot on a map generated by the Cartographer without glass detection (standard settings). The maps generated by the proposed *Cartographer_glass_lite* framework and the *slam_glass* algorithm are shown in Fig.4.9.

It can be observed that the map generated by the proposed cartographer improvement (*Cartographer_glass_lite*) is more accurate than that generated by the Gmapping based (*slam_glass*) map. Not only are the glass walls are not accurately mapped, the overall map has errors. This is clearly evident in the left side of the map where the map shows a misalignment.

Figure 4.4: Cartographer map with standard setting from *Building 1* with the glass walls manually marked in green

Figure 4.5: Default Cartographer map from *Building 1* with the robot path visualized in blue

Figure 4.7: Cartographer map with standard setting from *Building 2* overlayed with the floor plan with glass walls marked manually in green

(a) *Cartographer_glass_lite* map

(b) *slam_glass* map

Figure 4.6: Glass detection and mapping enabled maps generated from the data collected from *Building 1*. The spots which the proposed Cartographer based SLAM outperformed the *slam_glass* map are highlighted in red color where-as the spots which the *slam_glass* map had done better are highlighted in blue. The green marks indicate false points added in the *slam_glass* map.

Figure 4.8: Cartographer map with standard setting for *Building 2* with the robot path visualized

4.5.3 Evaluation of the results

From visual inspection, it is observed that the maps generated by the proposed solution based are more accurate than those generated by the GMapping based *slam_glass* algorithm. However, some undetected parts in the proposed SLAM maps are observed as well. These might be because the robot might not have had a normal incidence from those areas. This could have a few possible causes. The glass panels might not have been precisely vertical, or the ground might have had a slight inclination.

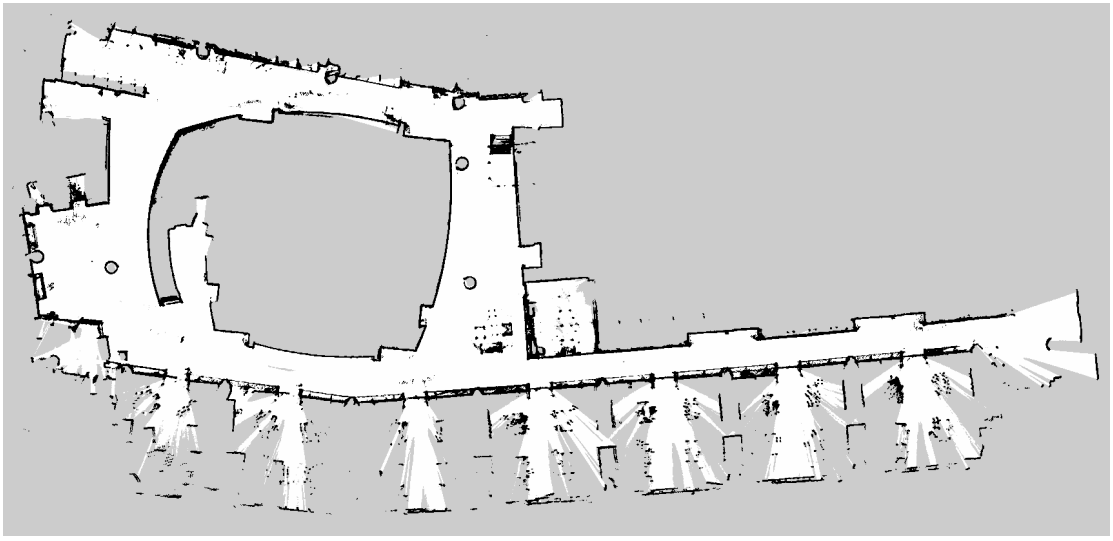
To evaluate the results, the lengths of glass walls that were detected by the proposed algorithm and the *slam_glass* algorithm are manually determined for comparison. Since the ground truth of the data set provided in [89] was not available to us, it is not included it in this evaluation. Table 4.2 shows the performance of the proposed glass detection and mapping algorithm. It is clearly seen that the proposed Graph SLAM based glass detection and mapping framework generates more accurate maps compared to the existing method.

Table 4.2: Performance comparison of glass mapping

Building	Glass length (m)	Detected glass (m)		Glass accuracy (%)	
		Cartographer	Gmappping	Cartographer	Gmapping
No.1	140.5	120.3	118.2	85.9	84.0
No.2	110.0	107.2	49.1	97.5	44.5

4.6 Discussion

A framework to incorporate glass detection and mapping capabilities to an optimization-based SLAM framework is introduced. The proposed algorithm was named *Cartographer_glass*.



(a) *Cartographer_glass_lite* map

(b) *slam_glass* map

Figure 4.9: Maps generated by the proposed *Cartographer_glass_lite* and from the *slam_glass* algorithm for data collected from *Building 2*

It is observed that the proposed algorithm's maps had better accuracy than those generated by the *slam_glass* map, which uses the same glass detection scheme but uses a particle filter based SLAM approach. Identifying heterogeneous material surfaces and labeling them accordingly in the generated map may be considered in the future. Further, accurate registration of glass objects in face of drift accumulation is also a challenge.

Chapter 5: Accurate registration of transparent objects in 2D LiDAR SLAM

5.1 Overview

In this chapter, the intensity spikes that occur when an incident beam strikes a transparent surface perpendicularly are exploited and an accurate account of such points is kept. Concurrent mapping of transparent objects suffers from the problem of *drift accumulation*. This is mitigated by utilizing the best trajectory estimate available from the Pose Graph Optimization (PGO) framework for mapping. This makes the the mapping of the detected points unaffected by the drift accumulated as the robot moves through the environment. This method exhibits superior results when compared to an existing approach that utilizes intensity spikes for detecting transparent surfaces with a particle filter based approach for generating map of the environment. It is showcased how, by leveraging the inherent structure of a PGO framework, the transparent objects can be mapped accurately.

In this chapter, the sharp spike in reflected beam intensity to detect the presence of glass and other transparent surfaces is exploited (introduced in [89]). As the robot moves through the environment these identified points are maintained in a separate data structure. It is showcased how these points can be mapped accurately onto the map of the environment by utilizing the history of poses maintained by the PGO framework. The maps produced by the proposed approach to those generated by the existing method [89] are compared. In the remainder of the

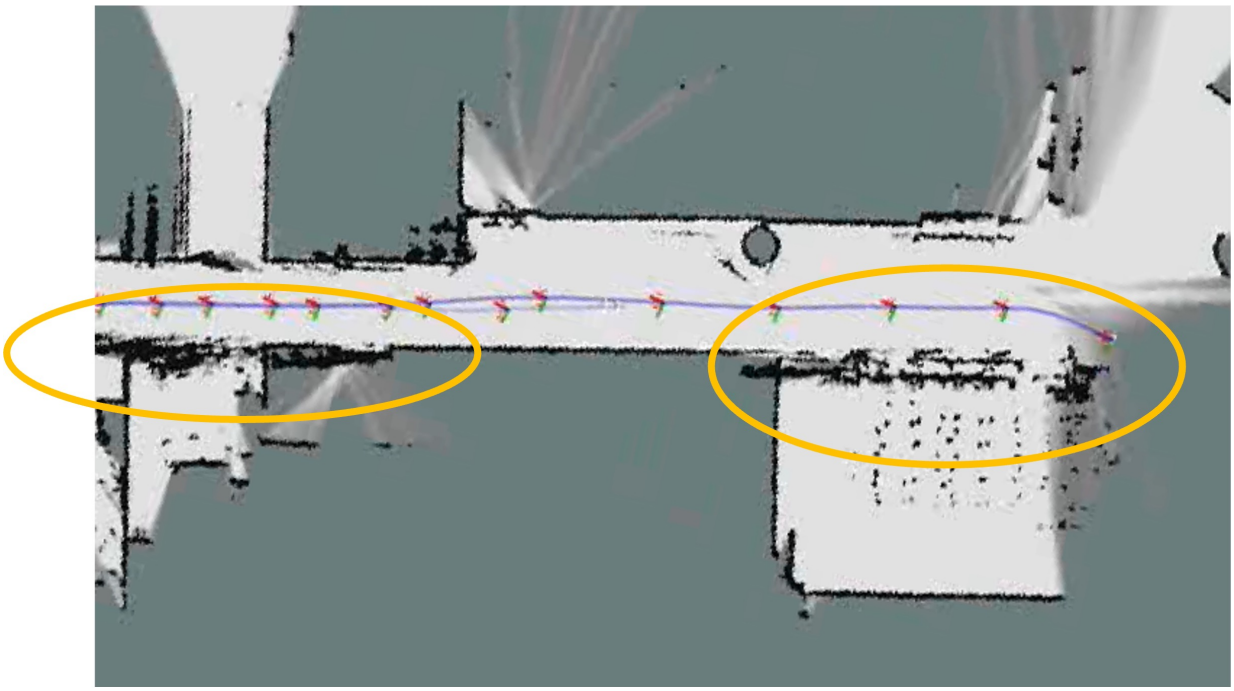


Figure 5.1: Drift accumulation causes same glass objects to appear distinct

chapter as previously, transparent *glass-like objects* and glass objects, will be referred to interchangeably.

5.2 Effects of drift on online mapping of transparent objects

In Chapter 4, transparent objects were mapped online as they were identified. The relative pose of scans in successive nodes of the pose graph is determined by the process of *scan matching*. This leads to an accumulation of error between the true pose of a node and its estimated pose. This is effect is illustrated in the Fig 5.1.

As the robot or the agent moves through the environment, it may revisit some parts of it. This presents an opportunity for the algorithm to minimize the accumulated drift by inserting a *loop closure constraint*. As a result, the pose of the robot along its trajectory are updated. Those objects that were mapped previously, when they are detected again, do not get mapped as the

same. The figure below shows a similar instance when the robot revisits a hallway it had been to previously. Same objects (marked in red) can be seen plotted as distinct.

5.3 Transparent Object Mapping via a Pose Graph Optimization Framework

The Pose Graph Optimization (PGO) framework [108], as the name suggests, consists of a graph where the nodes of the graph represent the poses of the robot. The edges between the nodes act as the constraints for the optimization problem. Successive nodes are connected by transformations that can be obtained by scan matching or by a reliable odometry source. Over time, drift may accumulate in the system and this results in the pose estimates becoming further from the true values. This drift can be corrected by incorporation of what are known as *loop closure* constraints. When the robot visits a portion of the environment that it had seen earlier, a *loop closure* constraint is inserted between the current node and a node from its previous visit. When the global optimization (consisting of all the poses and constraints thus far) is performed, the presence of *loop closure* constraints significantly reduces the drift. The proposed method is able to take advantage of this correction and able to deliver a consistent estimate of the glass points.

The glass detection methodology is incorporated along with a PGO framework. As a new LiDAR scan is received, it is either accepted or rejected by a motion detector based on whether the robot has moved or rotated beyond a certain threshold. This is done to avoid accumulation of redundant information. The accepted scan is then processed through a glass detection algorithm. If the scan contains any glass points, the indices of those points are recorded. This information equips us to find the local co-ordinates of the glass points with respect to the LiDAR via a

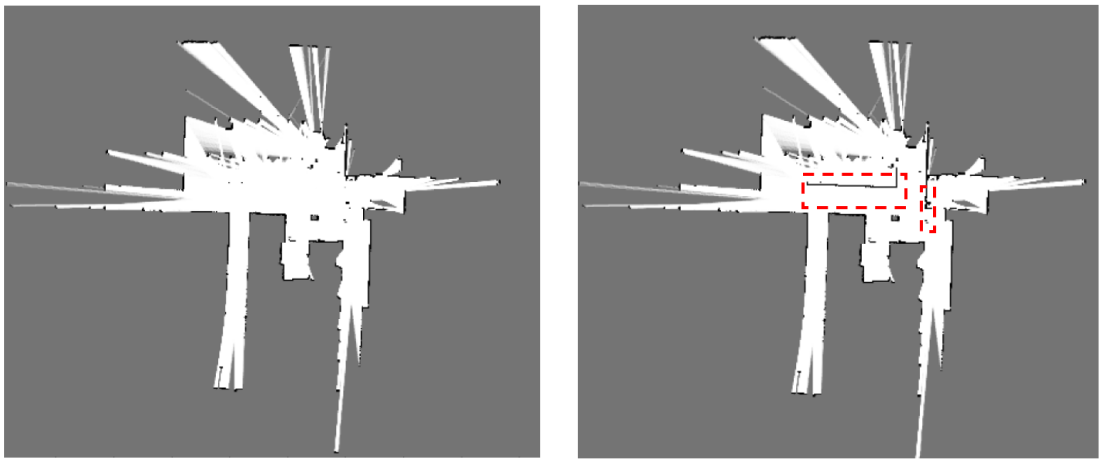
transformation of the polar co-ordinates (range and angle in the scan) to the local Cartesian co-ordinates in the LIDAR frame. If it is determined that a scan S has glass points at indices given by a set I, the local co-ordinates for each glass point $j \in I$ are obtained as $X_j^{local} = range_j \times cos(\theta_j)$ and $Y_j^{local} = range_j \times sin(\theta_j)$ where θ_j can be obtained by using the index of the point j in the scan and the angular resolution of the LiDAR.

A map of the environment, including the identified glass surfaces can be obtained at any time by utilizing the latest trajectory (poses of nodes) and the data structure containing the detected glass points. The global co-ordinates of the detected points can be recovered and plotted onto the map generated by the PGO framework. If a scan S is associated with a node N of the pose graph, and H represents its homogeneous transformation matrix with respect to the origin, the global co-ordinates of the glass point, $\{X_j^{global}, Y_j^{global}\}$ where $j \in I$, are obtained as follows:

$$\begin{bmatrix} X_j^{global} \\ Y_j^{global} \\ 1 \end{bmatrix} = H_j \begin{bmatrix} X_j^{local} \\ Y_j^{local} \\ 1 \end{bmatrix}$$

5.4 Results

The effectiveness of the proposed approach is demonstrated by applying it to different scenarios. In the first case, an off-the-shelf PGO framework from the MATLAB Navigation toolbox is applied to an open source dataset [89]. The results are compared to the same PGO framework augmented with the proposed solution. The parameters used include an intensity threshold of $thresh = 8000$, an intensity gradient of $grad = 4000$ and a profile width of $width =$



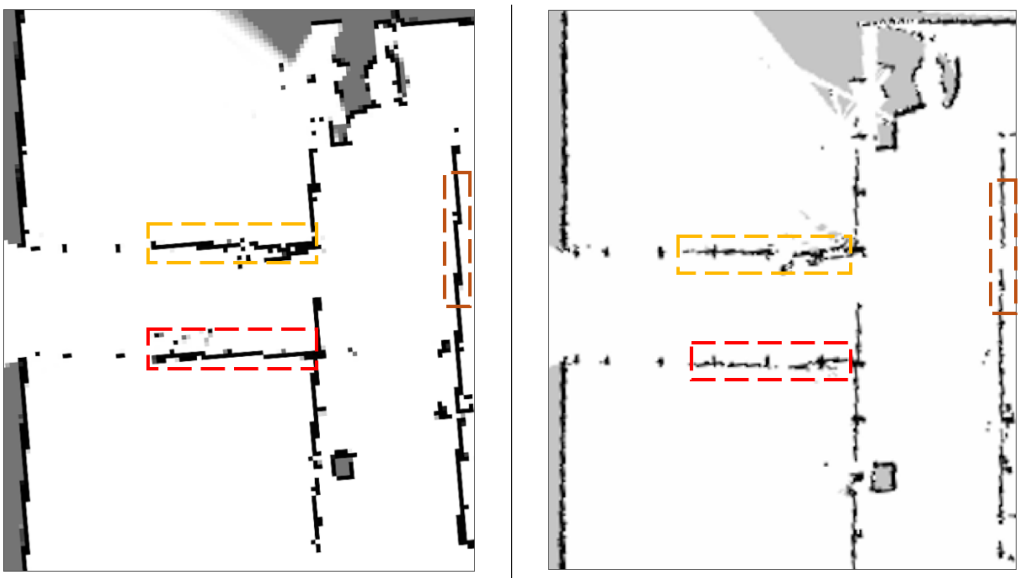
(a) PGO with no glass detection (b) PGO with glass detection

Figure 5.2: Comparison of maps generated by a PGO framework without and with glass detection where dashed boxes show glass surface undetected by the former

5 as suggested in [89]. The ability of the proposed method to successfully detect and add the transparent surfaces to the final map can be seen by comparing from Fig.5.2 (a) and Fig.5.2 (b) where the regions marked in dashed red in Fig.5.2 (a) are stretches of glass that are absent in Fig.5.2 (b).

In the second case, the performance of the augmented PGO framework with the method proposed in [89] is compared on a different segment of the open source dataset. The proposed method is able to recover a greater portion of the glass surface as shown in Fig. 5.3 when compared with the *slam_glass* algorithm [89].

For the case of large environments where the drift accumulated can be significant and the online mapping can result in errors in the map, the proposed method is able to perform the mapping accurately as showcased in Figure 5.4. The final occupancy grid in Figure 5.5 does not show any displacement between points detected on different visits.



(a) Proposed approach (b) *slam_glass* map

Figure 5.3: Generated maps for a part of the dataset from [89]. Dashed boxes in same color correspond to the same region.

5.5 Summary

This chapter presents a method for accurate registration of glass objects by utilizing the most recent trajectory estimate available from the Pose Graph Optimization (PGO) framework. The ability to handle the effect of drift by this method is showcased via experiments. However, the proposed glass detection method is contingent on the fact that near normal incidence results in spikes in reflected beam intensity. Also, the avenue of using reflected beam intensity for 3D LiDAR based glass detection should be explored.

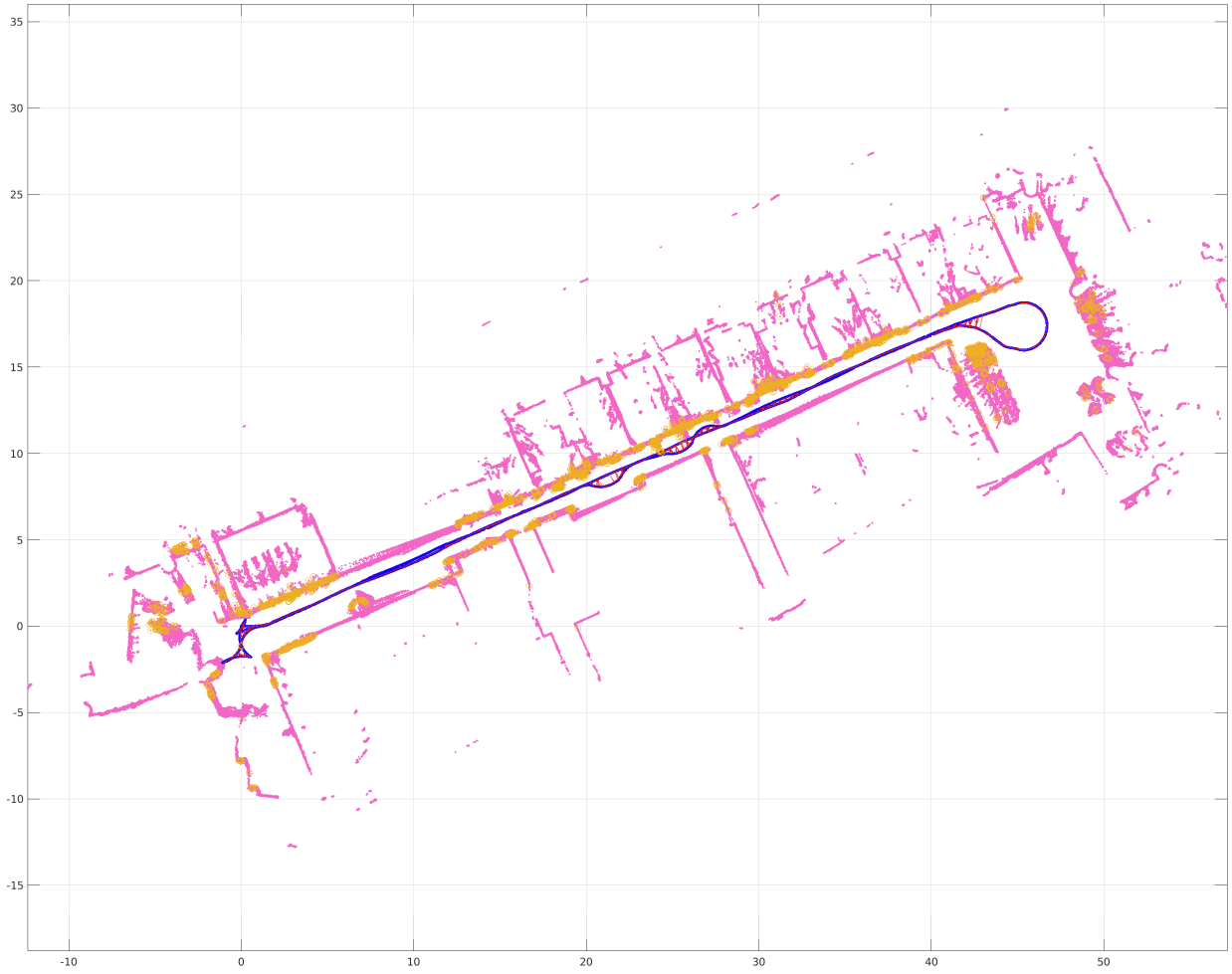


Figure 5.4: The trajectory of the robot is shown in blue, non-glass points in pink and the detected points in orange

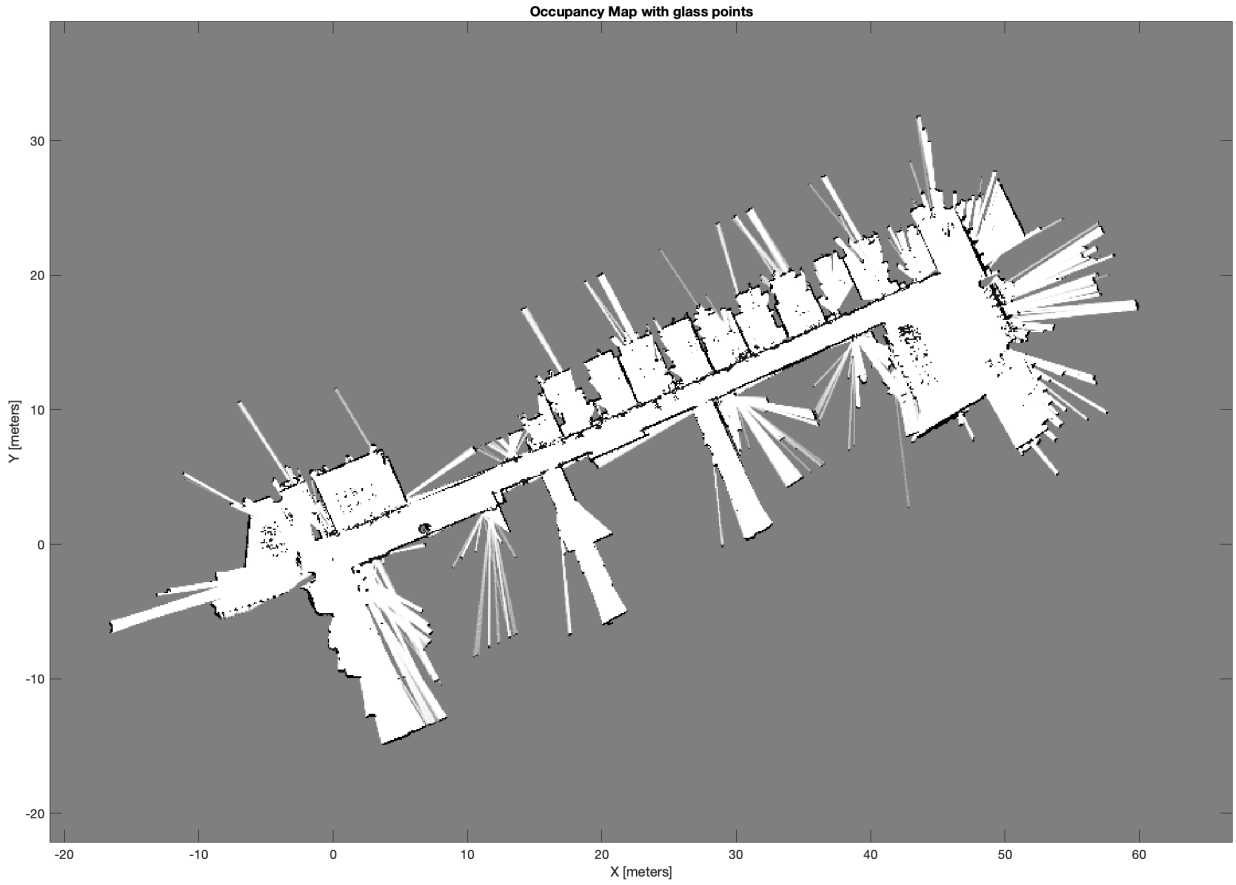


Figure 5.5: Final occupancy map

Chapter 6: Conclusions

6.1 Summary of contributions

This dissertation addresses the navigation of UGVs in complex environments by developing a supervised learning-based navigation algorithm that leverages MPC-based planner to provide high-quality labels for training the network in Chapter 2. This approach circumvents a major bottle neck around supervised learning models by automating the training data generation process. The deep learning-based planner is further enhanced with safety margins to improve its effectiveness in collision avoidance. The performance of the proposed method is then evaluated through simulations and real-world experiments, demonstrating its superior performance in terms of obstacle avoidance and successful mission completion.

In Chapter 3, data-based discovery of complex vehicle dynamics via universal differential equations is presented. It was showcased that via this method, the underlying dynamics can be learnt accurately and a computationally inexpensive method for incorporating the learnt dynamics into a model predictive control framework was presented. It was also highlighted that compared to other data-based contemporary approaches to this problem, the proposed method can be used to make predictions at different time scales.

Chapter 4 presented algorithms for detecting and mapping glass objects in a Graph SLAM framework. A simple and computationally inexpensive glass detection scheme was used to detect

glass objects. The efficacy of this approach via mapping of environments of different scales was showcased. In Chapter 5, the issue of *drift accumulation* that can affect mapping performance when operating in large environments is addressed.

6.2 Limitations and future research directions

The following is a list of some of the possible research directions that emerge from this dissertation,

1. *Time-varying dynamics modelling for UGVs* During the course of the operation of a vehicle, its properties may change due to wear and tear as well as part of its intended use. A UDE framework with fixed set of system weights is unable to account for changing system parameters.

2. *Dynamics modelling with noisy measurements and time delays* The work presented in Chapter 3 uses data collected via a simulator to model the unknown system dynamics. Real world systems have measurement noise and may require additional considerations for dealing with noisy data. It is known that the task of training UDEs becomes substantially harder with the increase in the dimensionality of the problem. As such the presence of time-delays can pose a challenge in using UDEs for learning the system dynamics.

3. *Testing of autonomous navigation solutions for indoor navigation* Since the scenarios presented while training a deep learning model for navigation might not cover the range of possibilities that the UGV might encounter during indoor operation, elaborate testing that

includes obstacles with varying speeds, different motion patterns and obstacle density (number of obstacles per unit area) should be investigated. Since the car-like robots can not turn in place, the solutions presented for differential drive robots may not apply to them.

4. *Transfer learning for adapting learnt dynamics to new UGV platforms* The model trained via the UDE-based approach can serve as a useful starting point for a vehicle that belongs to the same category (front-steered and rear wheel driven, for instance). Transfer learning can serve as a useful tool for achieving this objective.

5. *Improved localization using detected glass objects for loop closures* Since the glass objects are not visible from most viewing angles, the scan matching algorithm can not exploit these for loop closures. Enabling feature matching in the collection of glass points can help improve localization in environments with high presence of such objects.

Bibliography

[1] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Motion control for mobile robot navigation using machine learning: a survey. *arXiv preprint arXiv:2011.13112*, 2020.

[2] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[3] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[4] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

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

[6] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

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

[8] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505. IEEE, 1985.

[9] Maher Khatib and Raja Chatila. An extended potential field approach for mobile robot sensor-based motions. In *Proc. International Conference on Intelligent Autonomous Systems (IAS'4)*, 1995.

[10] Ahmad A Masoud. Decentralized self-organizing potential field-based control for individually motivated mobile agents in a cluttered environment: A vector-harmonic potential field approach. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(3):372–390, 2007.

[11] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[12] Christoph Rösmann. Difference between dwa and teb local planners.

[13] Chengmin Zhou, Bingding Huang, and Pasi Fränti. A review of motion planning algorithms for intelligent robots. *Journal of Intelligent Manufacturing*, pages 1–38, 2021.

[14] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, 17(7):760–772, 1998.

[15] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[16] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[17] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, 88:142–153, 2017.

[18] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, and Kurt Konolige. g2o: A general framework for (hyper) graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pages 9–13, 2011.

[19] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686. IEEE, 2017.

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

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

[22] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

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

[24] 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. In *2017 ieee international conference on robotics and automation (icra)*, pages 1527–1533. IEEE, 2017.

[25] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.

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

[27] Michael Everett, Yu Fan Chen, and Jonathan P How. Collision avoidance in pedestrian-rich environments with deep reinforcement learning. *IEEE Access*, 9:10357–10377, 2021.

[28] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

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

[31] Michael Everett, Yu Fan Chen, and Jonathan.P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. pages 3052–3059, 2018.

[32] Xuesu Xiao, Zizhao Wang, Zifan Xu, Bo Liu, Garrett Warnell, Gauraang Dhamankar, Anirudh Nair, and Peter Stone. Appl: Adaptive planner parameter learning. *Robotics and Autonomous Systems*, 154:104132, 2022.

[33] Xinjie Yao, Ji Zhang, and Jean Oh. Following social groups: Socially compliant autonomous navigation in dense crowds. *arXiv preprint arXiv:1911.12063*, 2019.

[34] Bruno Brito, Michael Everett, Jonathan P How, and Javier Alonso-Mora. Where to go next: learning a subgoal recommendation policy for navigation in dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):4616–4623, 2021.

[35] Ashwini Pokle, Roberto Martín-Martín, Patrick Goebel, Vincent Chow, Hans M Ewald, Junwei Yang, Zhenkai Wang, Amir Sadeghian, Dorsa Sadigh, Silvio Savarese, et al. Deep local trajectory replanning and control for robot navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5815–5822. IEEE, 2019.

[36] Wei Gao, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In *Conference on robot learning*, pages 185–194. PMLR, 2017.

[37] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. Toward agile maneuvers in highly constrained spaces: Learning from hallucination. *IEEE Robotics and Automation Letters*, 6(2):1503–1510, 2021.

[38] Mark Pfeiffer, Samarth Shukla, Matteo Turchetta, Cesar Cadena, Andreas Krause, Roland Siegwart, and Juan Nieto. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4423–4430, 2018.

[39] Jingyun Ning and Madhur Behl. Scalable deep kernel gaussian process for vehicle dynamics in autonomous racing. In *Conference on Robot Learning*, pages 761–773. PMLR, 2023.

[40] Achin Jain, Matthew O'Kelly, Pratik Chaudhari, and Manfred Morari. Bayesrace: Learning to race autonomously using prior experience. *arXiv preprint arXiv:2005.04755*, 2020.

[41] Ugo Rosolia and Francesco Borrelli. Learning how to autonomously race a car: a predictive control approach. *IEEE Transactions on Control Systems Technology*, 28(6):2713–2719, 2019.

[42] Haoru Xue, Edward L Zhu, and Francesco Borrelli. Learning model predictive control with error dynamics regression for autonomous racing. *arXiv preprint arXiv:2309.10716*, 2023.

[43] Benjamin Van Niekerk, Andreas Damianou, and Benjamin Rosman. Online constrained model-based reinforcement learning. *arXiv preprint arXiv:2004.03499*, 2020.

[44] Pranav Atreya, Haresh Karnan, Kavan Singh Sikand, Xuesu Xiao, Sadegh Rabiee, and Joydeep Biswas. High-speed accurate robot control using learned forward kinodynamics and non-linear least squares optimization. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11789–11795. IEEE, 2022.

[45] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[46] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[47] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[48] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[49] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2432–2437. IEEE, 2005.

[50] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[51] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[52] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[53] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm intelligence*, 2(2):189–208, 2008.

[54] Gurtajbir Herr, Lasitha Weerakoon, Miao Yu, and Nikhil Chopra. Cardynet: Deep learning based navigation for car-like robots in dynamic environments. In *ASME International Mechanical Engineering Congress and Exposition*, volume 86670, page V005T07A074. American Society of Mechanical Engineers, 2022.

[55] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Distributed autonomous robotic systems*, pages 203–216. Springer, 2013.

[56] Min Gyu Park, Jae Hyun Jeon, and Min Cheol Lee. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. In *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, volume 3, pages 1530–1535. IEEE, 2001.

[57] Jing Liang, Utsav Patel, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Realtime collision avoidance for mobile robots in dense crowds using implicit multi-sensor fusion and deep reinforcement learning. *arXiv preprint arXiv:2004.03089*, 2020.

[58] Peter Regier, Lukas Gesing, and Maren Bennewitz. Deep reinforcement learning for navigation in cluttered environments. In *Proc. of the Intl. Conf. on Machine Learning and Applications (CMLA)*, 2020.

[59] Heechan Shin, Donghyuk Kim, and Sung-Eui Yoon. Kinodynamic comfort trajectory planning for car-like robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6532–6539. IEEE, 2018.

[60] Tomasz Gawron and Maciej Marcin Michalek. Algorithmization of constrained motion for car-like robots using the vfo control strategy with parallelized planning of admissible funnels. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6945–6951. IEEE, 2018.

[61] Jacob J Johnson, Linjun Li, Fei Liu, Ahmed H Qureshi, and Michael C Yip. Dynamically constrained motion planning networks for non-holonomic robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6937–6943. IEEE, 2020.

[62] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt*. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483, 2011.

[63] Pablo Marin-Plaza, Ahmed Hussein, David Martin, and Arturo de la Escalera. Global and local path planning study in a ros-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, 2018, 2018.

[64] Xuehao Sun, Shuchao Deng, Tingting Zhao, and Baohong Tong. Motion planning approach for car-like robots in unstructured scenario. *Transactions of the Institute of Measurement and Control*, page 0142331221994393, 2021.

[65] Franz Albers, Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Online trajectory optimization and navigation in dynamic environments in ros. In *Robot Operating System (ROS)*, pages 241–274. Springer, 2019.

[66] Tonja Heinemann, Moritz Villinger, Armin Lechler, and Alexander Verl. Dynamic obstacle layer: A new concept to avoid moving obstacles in local path planning using ros. In *ISR 2020; 52th International Symposium on Robotics*, pages 1–8, 2020.

[67] B. Cybulski, A. Wegierska, and G. Granosik. Accuracy comparison of navigation local planners on ros-based mobile robot. In *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, pages 104–111, 2019.

[68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[69] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[70] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

[71] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[72] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[73] Marcell Missura and Maren Bennewitz. Predictive collision avoidance for the dynamic window approach. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8620–8626. IEEE, 2019.

[74] S Russell and Peter Norvig. Solving problems by searching. *Artificial intelligence: a modern approach*, pages 64–119, 1995.

[75] Francois Chollet et al. Keras, 2015.

[76] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

[77] Alexander Liniger. *Path planning and control for autonomous racing*. ETH Zurich, 2018.

[78] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[79] Chang Mook Kang, Seung-Hi Lee, and Chung Choo Chung. Comparative evaluation of dynamic and kinematic vehicle models. In *53rd IEEE Conference on Decision and Control*, pages 648–653. IEEE, 2014.

[80] Jingyun Ning and Madhur Behl. Vehicle dynamics modeling for autonomous racing using gaussian processes. *arXiv preprint arXiv:2306.03405*, 2023.

[81] Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

[82] Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research*, 123, 2020.

[83] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[84] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[85] Christopher Rackauckas and Qing Nie. DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*, 5(1), 2017.

[86] 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.

[87] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

[88] Paul Foster, Zhenghong Sun, Jong Jin Park, and Benjamin Kuipers. Visagge: Visible angle grid for glass environments. In *2013 IEEE International Conference on Robotics and Automation*, pages 2213–2220. IEEE, 2013.

[89] Xun Wang and JianGuo Wang. Detecting glass in simultaneous localisation and mapping. *Robotics and Autonomous Systems*, 88:97 – 103, 2017.

[90] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[91] Rainer Koch, Stefan May, Philipp Koch, Markus Kühn, and Andreas Nüchter. Detection of specular reflections in range measurements for faultless robotic slam. In *Robot 2015: Second Iberian Robotics Conference*, pages 133–145. Springer, 2016.

[92] Rainer Koch, Stefan May, Patrick Murmann, and Andreas Nüchter. Identification of transparent and specular reflective material in laser scans to discriminate affected measurements for faultless robotic slam. *Robotics and Autonomous Systems*, 87:296–312, 2017.

[93] Jiwoong Kim and Woojin Chung. Robust localization of mobile robots considering reliability of lidar measurements. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6491–6496. IEEE, 2018.

[94] Muhammad Awais. Improved laser-based navigation for mobile robots. In *2009 International Conference on Advanced Robotics*, pages 1–6. IEEE, 2009.

[95] Jun Jiang, Renato Miyagusuku, Atsushi Yamashita, and Hajime Asama. Glass confidence maps building based on neural networks using laser range-finders for mobile robots. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 405–410. IEEE, 2017.

[96] Jun Jiang, Renato Miyagusuku, Atsushi Yamashita, and Hajime Asama. Robust map registration for building online glass confidence maps. *IFAC-PapersOnLine*, 52(8):362–367, 2019.

[97] Jin-Woo Jung, Jung-Soo Park, Tae-Won Kang, Jin-Gu Kang, and Hyun-Wook Kang. Mobile robot path planning using a laser range finder for environments with transparent obstacles. *Applied Sciences*, 10(8):2799, 2020.

[98] Jie Meng, Shuting Wang, Gen Li, Liquan Jiang, Yuanlong Xie, and Chao Liu. Accurate lidar-based localization in glass-walled environment. In *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 534–539. IEEE, 2020.

[99] Zhiming Huang, Kaiwei Wang, Kailun Yang, Ruiqi Cheng, and Jian Bai. Glass detection and recognition based on the fusion of ultrasonic sensor and rgb-d sensor for the visually impaired. In *Target and Background Signatures IV*, volume 10794, page 107940F. International Society for Optics and Photonics, 2018.

[100] Albert Diosi and Lindsay Kleeman. Advanced sonar and laser range finder fusion for simultaneous localization and mapping. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1854–1859. IEEE, 2004.

[101] Hao Wei, Xueen Li, Ying Shi, Bo You, and Yi Xu. Fusing sonars and lrf data to glass detection for robotics navigation. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 826–831. IEEE, 2018.

[102] Hao Wei, Xue-en Li, Ying Shi, Bo You, and Yi Xu. Multi-sensor fusion glass detection for robot navigation and mapping. In *2018 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pages 184–188. IEEE, 2018.

[103] Shao-Wen Yang and Chieh-Chih Wang. Dealing with laser scanner failure: Mirrors and windows. In *2008 IEEE International Conference on Robotics and Automation*, pages 3009–3015. IEEE, 2008.

[104] S. Yang and C. Wang. On solving mirror reflection in lidar sensing. *IEEE/ASME Transactions on Mechatronics*, 16(2):255–265, 2011.

[105] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[106] Daniel Wilbers, Christian Merfels, and Cyrill Stachniss. A comparison of particle filter and graph-based optimization for localization with landmarks in automated vehicles. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 220–225. IEEE, 2019.

[107] Kirill Krinkin, Anton Filatov, Art yom Filatov, Artur Huletski, and Dmitriy Kartashov. Evaluation of modern laser based indoor slam algorithms. In *2018 22nd Conference of Open Innovations Association (FRUCT)*, pages 101–106. IEEE, 2018.

[108] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[109] Sertac Karaman, Ariel Anders, Michael Boulet, Jane Connor, Kenneth Gregson, Winter Guerra, Owen Guldner, Mubarik Mohamoud, Brian Plancher, Robert Shin, et al. Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit. In *2017 IEEE integrated STEM education conference (ISEC)*, pages 195–203. IEEE, 2017.