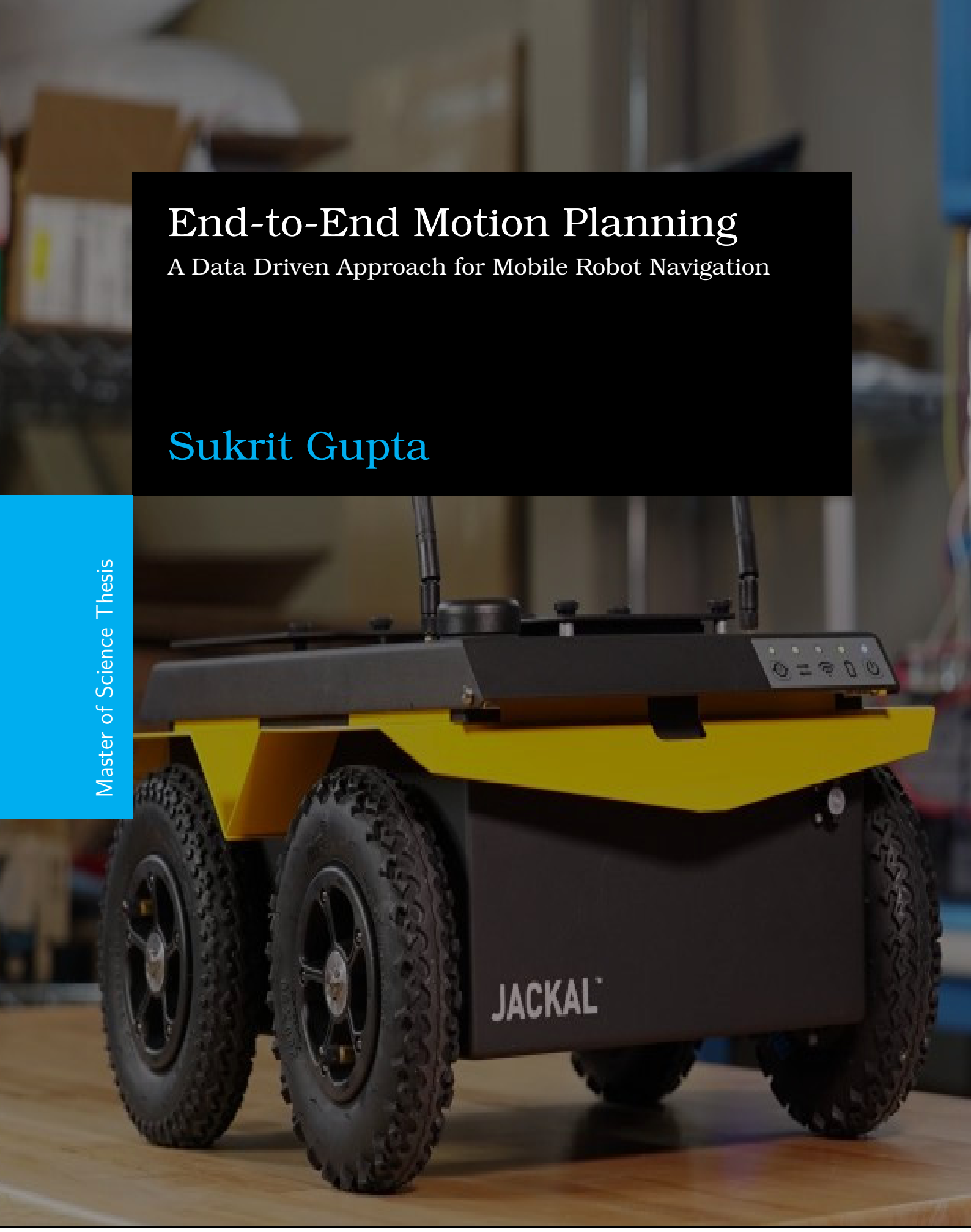
End-to-End Motion Planning

A Data Driven Approach for Mobile Robot Navigation

Sukrit Gupta

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control | Cognitive Robotics

End-to-End Motion Planning

A Data Driven Approach for Mobile Robot Navigation

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Sukrit Gupta

February 24, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

A lot research has been conducted in the field of autonomous navigation of mobile robots with focus on Robot Vision and Robot Motion Planning. However, most of the classical navigation solutions require several steps of data pre-processing and hand tuning of parameters, with separate modules for vision, localization, planning and control. All these modules work independently and make their own parameter assumptions to optimize their own performance without taking into account the effect these assumptions have on the performance of rest of the modules. Hence, even though each module in the whole system tries to achieve an optimal performance for the task it has been assigned, the lack of interdependence exhibited by these modules for decision making means that the overall performance of the whole system is sub-optimal in most of the cases. An alternating approach for addressing these issues is to train certain parts of the vision module to incorporate partial tasks from the planning module.

Deep Learning architectures have achieved great success in the field of pattern recognition and object detection and as a result are usually being deployed to design such a module that jointly learns to carry out perception and path planning. This master's thesis, making use of Deep Learning, proposes an End-to-End Learning architecture that learns to directly map raw sensor readings to control commands for a ground based mobile robot. The research makes use of the simulation of Jackal UGV from Clearpath Robotics and the proposed network is able to produce collision free trajectories for the robot to navigate in it's environment.

Table of Contents

List of Figures

List of Tables

Acknowledgements

The work presented in thesis is a culmination of two and a half years of journey filled with numerous ups and downs, joys and sorrows and a whole lot of Robotics. This journey, however will be incomplete without thanking the people who helped me reach the finish line.

My thesis supervisor, **Dr. Javier Alonso Mora**, thank you for giving me the opportunity to work with your highly competitive and amazing research group. It was a great learning experience where I got to know a lot about scientific research. After coming to Delft, my main aim was to conduct research in the field of Robotics and you helped me achieve that objective. I am also grateful to you for giving me various opportunities to present demos of our group's research work. They were a great learning experience and helped me a lot with my confidence.

My daily supervisor, **Bruno Ferreira de Brito**, I can't thank you enough. Right through the beginning of my thesis, you have been highly supportive through all my ups and downs and have backed me in the best possible way. You have always been there to help me with various doubts, no matter how busy you are. Your work ethic is something I inspire to achieve. Thank you for guiding me through my first ever experience of scientific research and helping me improve my programming skills.

Neel Nagda, I couldn't have asked for a better friend (read as brother) on this gruelling journey. I will cherish all our team-ups for various assignments and thanks for always having my back. A big thank you to my **parents**. You guys have been the biggest strength of my life and even though you have been thousands of kilometers away for these last couple of years, your support and understanding for helping me realise my dreams has been immense. Lastly, I would like to dedicate this research work to my late **grandmother**, who always loved and cared for me and dreamt for me to achieve great things in life.

Now that you, the reader have managed to reach the end of this page, I would like to thank you for showing interest in my work and I invite you to continue reading.

Delft, University of Technology — Sukrit Gupta
February 24, 2020

Chapter 1

Introduction

1-1 Motivation

Making the robots operate as desired by it's operator is one of major challenges in the field of Robotics. For a ground based mobile robot, this challenge is defined as navigating safely from the current position to a goal position using a control policy. A lot research has been conducted in this field with focus on Robot Vision and Robot Motion Planning. However, most of the classical navigation solutions require several steps of data preprocessing and hand tuning of parameters, with dedicated modules for vision, localization, planning and control. A map of the environment has to be provided, relevant features need to be extracted from the sensor data, and obstacles have to be detected. All these modules work independently to optimize their own performance without taking into account the effect these optimizations may have on the performance of rest of the modules. Hence, even though each module in the whole system tries to achieve an optimal performance for the task it has been assigned, the lack of interdependence exhibited by these modules for decision making means that the overall performance of the whole system is sub-optimal in most of the cases. An alternating approach for addressing these issues is to train certain parts of the vision module to incorporate partial tasks from the planning module.

With the recent advancement of the Graphics Processing Unit (GPU) and availability of huge number of large datasets, Deep Learning has made a huge progress in recent years. Deep Learning architectures like Convolution Neural Networks (CNN) have particularly achieved a tremendous success in the field of pattern recognition and object detection due to their ability to learn their own features from the raw data and as a result are usually being deployed to design such a module that jointly learns to carry out perception and path planning. This master's thesis aims to leverage on such Deep Learning techniques to create an End-to-End Learning architecture that learns an entire processing pipeline by combining the vision and motion planning modules to steer a mobile robot through it's surrounding environment.

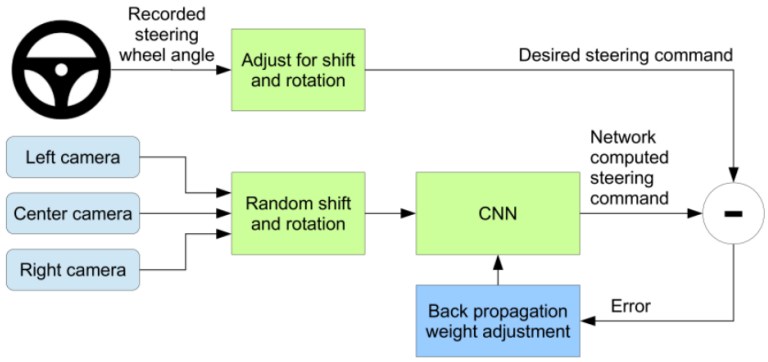


Figure 1-1: Block diagram of system proposed in NVIDIA's DAVE 2 [1]

1-2 Related Work

A significant amount of work in End-to-End navigation has focused on autonomous driving on highways and local roads. NVIDIA [1] trained a CNN to map raw pixels from a single front-facing camera directly to steering commands. With minimum training data, the system learns to drive in traffic on local roads with or without lane markings and on highways. With only human steering angle as training signal, the system is able to learn useful road features during various internal processing steps. The training data for the network is augmented with additional images containing different shifts and rotations to help the network learn to recover from mistakes (see Figure 1-1). [3] describes how the network proposed in [1] makes its decisions. It discusses the method for determining which elements in the road image most influence the system's steering decision. Results show that the system indeed learns to recognize relevant objects on the road. Alongside learning general feature such as lane markings, other cars and edges of the roads, the system also learns features like bushes lining the edge of the road, which are quite hard to program.

[4] integrates Light Detection and Ranging (LiDAR) point clouds, Google driving directions and GPS-IMU data to generate driving paths. The system consists of fully convolutional neural network that learns perception and path generation from real-world driving data. The approach also avoids the time consuming and expensive hand labeling of training data as the data is automatically collected in the form of steering angle logs and their corresponding time stamped sensor readings. But one of the biggest drawbacks of this approach is that it involves a number of data preprocessing steps that that adversely affects the real-time implementation of the network. [2] presents a model that is able to learn mapping from 2D laser- range findings and a target position to the desired steering commands for the robot. To train the motion planner, the data is generated using Robot Operating System (ROS) Move Base that makes use of Dijkstra algorithm as the global planner and the Dynamic Window Approach as the local planner in a simulated environment. A 270 degree laser range finder is used as the only sensor of the robot for spacial scene understanding and collision avoidance. The designed deep planner is depicted in Figure 1-2. Some of the major drawbacks of this model are that the robot heading frequently fluctuates while navigating and robot occasionally gets stuck around the obstacles resulting in joystick interventions.

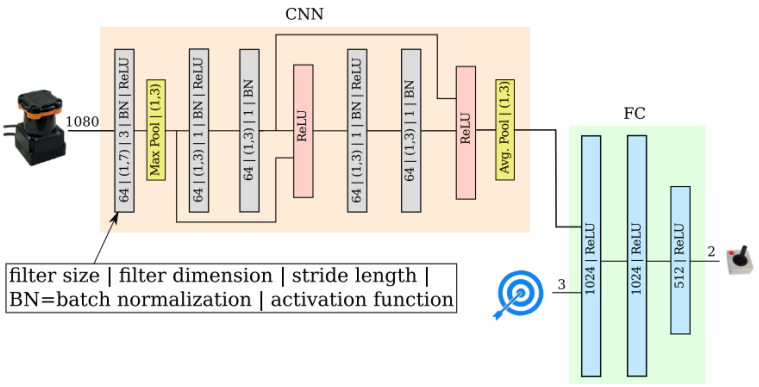


Figure 1-2: Deep Planner Netowrk [2]

[5] introduces the concept of an iterative motion planning algorithm called Motion Planning Network (MPNet) which is a neural network based planning algorithm that generates end-to-end collision-free paths irrespective of the obstacle's geometry. It consists of a Contractive Autoencoder and a deep feedforward neural network. The autoencoder encodes the workspace from the point cloud data and the deep network takes this workspace encoding along with the start and goal configuration to generate end-to-end feasible trajectories for a mobile robot to follow. The deep feedforward neural network part of the model is trained using Rapidly exploring Random Trees* (RRT*) to generate feasible, near optimal paths in various environments. The proposed model along with being computationally efficient in 2D and 3D environments also generalizes quite well on completely unseen static environments.

[6] introduces a reinforcement learning method for exploring a corridor environment with the depth information from an Red Green Blue - Depth (RGB-D) sensor only. The robot controller pre-trains the feature maps using the depth information and achieves obstacle avoidance. The system uses Deep Q-Network (DQN) framework for the Q-value estimation of the Q-learning method. The controller is implemented in two steps, a supervised deep learning structure and a Q-learning network. The supervised learning model takes the depth information as input and the command as output. This supervised learning model is implemented with a Convolution Neural Network and the feature maps from the second last layer of this network are used as input by the Q-learning network, which contains three fully-connected hidden layers. The experiments are conducted on a Turtlebot in the Gazebo simulation environment and the controller shows robustness to different kinds of corridor environments. Similarly, [7] overcomes the need of time consuming building and updating of obstacle map by designing a learning based mapless motion planner that takes the sparse 10-dimensional range findings and the target position with respect to the mobile robot coordinate frame as input and the continuous steering commands as output. The motion planner is trained end-to-end via asynchronous deep reinforcement learning and is applied to a non holonomic differential drive platform in unseen virtual and real environments. [8] proposes an actor-critic model for target driven visual navigation. The policy of this model is a function of the goal as well as the current state, allowing for better generalization. The model finds the minimum length sequence of actions that move an agent from its current location to a target that is specified by an RGB image. It takes as input a RGB image of the current observation and another

RGB image of the target and learns a mapping from the 2D image to an action in the 3D space such as move forward or turn right.

Due to a heavy reliance on learning from trial and error, Reinforcement Learning methods require a large amount of time for system interaction. Plus, a well designed reward function is necessary to find an optimal policy. Imitation learning, as an alternative to learning control policies, guides the policy search, not by hand-designed reward signals, but by providing the learning agent with expert's demonstrations [9]. It enables the agents to learn successful policies in fields where people can easily demonstrate the desired behavior but find it difficult to hand program or hard code the correct cost or reward function. This is especially useful for robots with high degrees of freedom. [10] introduces and end-to-end imitation learning system for off-road autonomous driving using low cost on-board sensors. The proposed system uses a Model Predictive Control (MPC) as an expert to train a deep neural net in order to map raw, high- dimensional data to continuous steering and throttle commands. The system neither requires a state estimation nor an on-the-fly planning to navigate the vehicle. The policies trained on the system with online imitation learning generalize quite well and overcome challenges related to covariate shift.

In supervised learning, an end-to-end autonomous driving policy function is tuned to minimize the difference between the predicted and ground-truth actions. A policy function trained this way can exhibit undesirable behaviour due to mismatch between the states that can be reached by the reference policy and the trained policy. Imitation learning algorithms like, DAgger [11] address this issue by iteratively collecting training samples from both reference and trained policy. But, these algorithms often require a large number of queries to the reference policy which is quite expensive. [12] overcomes this issue by proposing an extension of DAgger, called SafeDAgger, which is query efficient and suitable for end-to-end autonomous driving. SafeDAgger achieves a superior performance compared to DAgger in terms of the time taken to converge to an optima and the average number of laps without crash and the amount of damage in simulated environment.

1-3 Problem Formulation

Humans have the ability to perceive their surroundings, extract relevant information and take decisions accordingly. In order to make those decision, they rely on a large set of experiences acquired throughout their lifetime. For mobile robots, in order to make such decisions like moving to a desired goal position, they have to overcome several hurdles. Firstly, the robot has to extract necessary information from the input sensor data, then has to obtain a model that maps the sensor readings to desired control commands and lastly during deployment, this model is required to make right decisions for each new input data. The aim of this thesis is to combine all these steps and directly compute control commands based on the sensor data. Given expert demonstrations, an End-to-End planner tries to learn the following mapping:

$$u = F_\theta(y, g)$$

that directly maps the sensor data y and goal information g to required control commands u. During network training, the best possible set of parameters θ are learnt that capture the

relation between the input-output pairs of the training data.

1-4 Research Objectives

The following research questions have been adopted in order to achieve the desired goals of this master's thesis:

- Can an End-to-End motion planner learn to navigate to a desired goal position by just using the raw sensor reading from the robot's local environment?
- Is a Supervised Learning based approach sufficient enough to capture the desired navigation behaviour?
- Does combining the Supervised Learning based approach with another learning methodology like Reinforcement Learning help in improving the performance of the proposed End-to-End planner?

1-5 Overview of the Thesis

The overview of the thesis is as follows:

- Chapter 2 presents and describes the proposed End-to-End network structure
- Chapter 3 details the procedure for the collection of the data, required to train the End-to-End network using Supervised Learning
- Chapter 4 Presents and discusses the performance results of the Supervised Learning based approach and establishes comparisons with other benchmarks
- Chapter 5 Details the methodology for a possible combination of Supervised Learning with Reinforcement Learning
- Chapter 6 Discusses the results of the End-to-End planner trained using a combination of different learning techniques.
- Chapter 7 concludes the report with suggestions for future work.

Chapter 2

End-to-End Network Architecture

This chapter proposes an End-to-End network that can learn the desired robot navigation behaviour. A description for every part of the network along with their role in robot navigation is provided. The proposed End-to-End network is presented in Figure 2-1.

2-1 The Input Channel

The first half of the proposed End-to-End network is the input channel, where different inputs to the network are processed/encoded accordingly to be further processed by the second half of the network. The input channel is further divided into two sub channels, one of which is responsible for encoding laser scan readings while the other handles the data regarding the goal information input to the network.

2-1-1 The Laser Scan Channel

The first sub input channel takes as input a vector of 2D Laser Scan readings and extracts relevant information from it. The input vector has a size of 1x720, where each of the 720 readings represents the distance to a detected obstacle from the mobile robot at a certain angle in meters. These readings are recorded with a field of view of 360 degrees. The information extraction in this sub input channel takes place in two stages. In the first stage, the input laser data is encoded to a vector of size 1x128 and in the second stage, the temporal relation between consecutive laser data encodings is learnt.

Encoding the Laser Scan Data

In order to encode the laser data, various operations are performed which are detailed as follows:

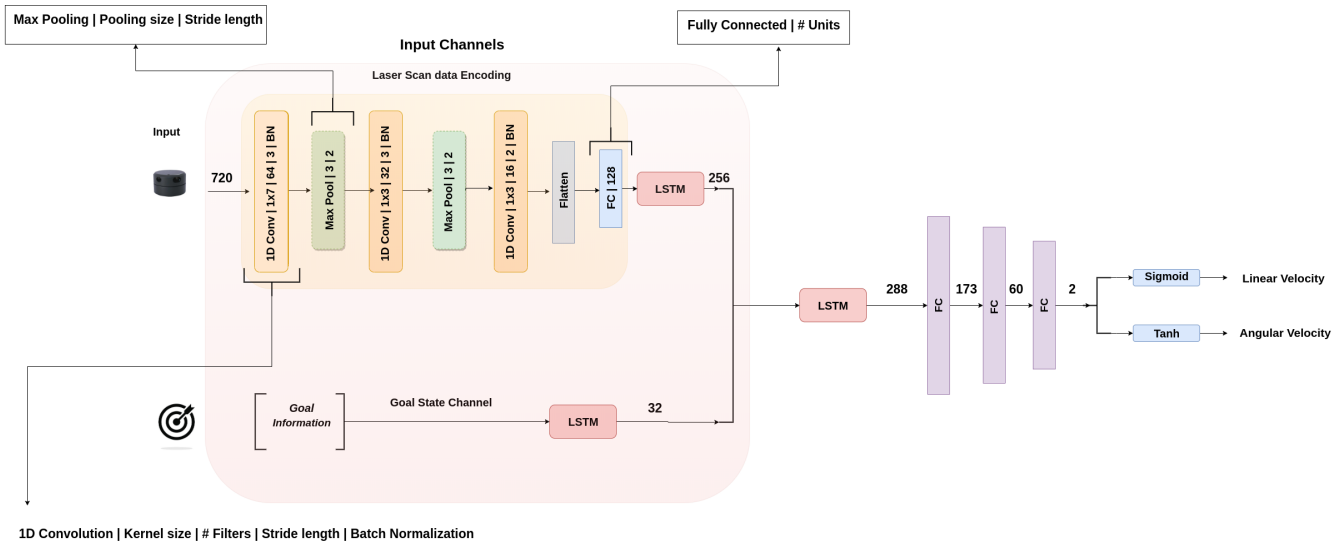


Figure 2-1: Model with input channels consisting of laser scan data and goal information

- **1D Convolutions**: The convolution layer uses filters of size F that scans the input of size I using strides of length S to obtain a feature map of size O. The input is also padded with P zeros before convolution in order to make the output of the convolution operation mathematically convenient. The padding operation uses the 'same' mode in the proposed network. This results in the following set of equations based on [13]:

$$\begin{aligned} P &= \frac{F - S}{2} \\ O &= \frac{I - F + 2P}{S} + 1 \end{aligned} \tag{2-1}$$

- **Batch Normalization**: In a neural network, as the parameters of the preceding layer change, the distribution of the input to the current layer changes, which results in the current layer constantly adjusting to these new distributions. As the network goes deeper, the impact of these distribution changes gets amplified. The application of Batch Normalization to the values obtained from convolution operation helps solve this issue [14]. This operation considers the mean μ_B and variance σ_B^2 of the current batch B of inputs and transforms each input x_i in the batch as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{2-2}$$

  Parameters, γ and β scale and shift the transformation respectively and help maintain stability of the network. These parameters are learnt during the network optimization step. Hyper-parameter, ϵ is added to the denominator for numerical stability and its value is set to 0.001. The application of Batch Normalization enables a faster training of the network and allows each layer of the network to learn by itself a bit more independently of the other layers.

- **Max Pooling**: The max pooling operation down samples an input by selecting the maximum from the current set of values in the view.

Table 2-1: Summary of operations for encoding laser data (NA = Not Applicable)

Operation	Kernel Size	#Filters	Strides	Batch Normalization	ReLu	Output size
1D Convolution	1 x 7	64	3	yes	yes	240/filter
Max Pool	1 x 3	NA	2	NA	NA	120/filter
1D Convolution	1 x 3	32	3	yes	yes	40/filter
Max Pool	1 x 3	NA	2	NA	NA	20/filter
1D Convolution	1 x 3	16	2	yes	yes	10/filter
Flatten	NA	NA	NA	NA	NA	160
Fully Connected	NA	NA	NA	no	yes	128

- **Fully Connected Layer**: The Fully Connected layer outputs the final encoding of size 1x128 by operating on the flattened output from the previous layers. Here each value in the flattened input is connected to all the neurons in the fully connected layer. The output of the fully connected layer, $\hat{\mathbf{y}}$ with input vector $\mathbf{x}$, is obtained using following equation:

$$\hat{\mathbf{y}} = \mathbf{w}^T\mathbf{x} + \mathbf{b} \tag{2-3}$$

  $\mathbf{w}$ and $\mathbf{b}$ are weights and biases respectively that are learnt during the network optimization step.

- **Rectified Linear Unit (ReLu)**: **Relu!** (**Relu!**) is an activation function g which is applied to the batch normalized outputs of the 1D convolution layers and the output of the fully connected layer. The activation function is used in order to introduce a non-linearity to the neural network. The ReLu activation is described by the following equation:

$$g(z) = max(0, z) \tag{2-4}$$

The above mentioned operations, performed by the network to encode the laser data are summarized in Table 2-1.

With the successful derivation of the laser data encoding, the next step is to learn a temporal relation between a sequence of these encodings.

Temporal Relation Between Laser Scan Encodings

Majority of the Supervised Learning approaches assume the input-output pairs of training data to be Independent and Identically Distributed (i.i.d). But in reality, when a mobile robot navigates, it's current control action directly affects the future sensor readings, it receives from it's surrounding environment. Thus, training an end-to-end robot motion planner with i.i.d assumption can lead to a highly unstable performance. As a result, the proposed network is required to have the ability to learn the temporal behaviour of the robot's control commands and the sensor input it receives from the environment. In the Laser Scan channel, the temporal behaviour of the laser readings is learnt with the use of a Long Short Term Memory (LSTM) cell.

The LSTM cell consists of a cell state and three gates, namely forget, input and output gates. The cell state retains information over a time period and the three gates regulate the flow of this information through the cell.

The first step for the LSTM cell in the proposed network is to decide how much of the laser scan information available at time $t-1$ is to be discarded from the cell state at time t. This decision is made by the sigmoid layer, σ of the 'forget gate', f_t. The forget gate considers h_{t-1}, the LSTM cell output at time $t-1$ and x_t, the laser scan information input to the cell at time t to output a value between 0 and 1 for each number in the cell state, C_{t-1}. A value of 1 completely retains the information and 0 discards all of it.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2-5}$$

The next step for the LSTM cell is to decide how much of the new laser scan information is going to be stored in the cell state, C_t. The sigmoid layer of the 'input gate', i_t decides the values that will be updated and a tanh layer provides a vector of new candidate values, $\tilde{C}_t$, that can be added to the cell state.

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= tanh(W_C[h_{t-1}, x_t] + b_C) \end{aligned} \tag{2-6}$$

Based on the equations presented above, the cell state can be updated as,

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2-7}$$

With the cell state updated, the final step is to output the relevant temporal information extracted from the laser scan encoding at time t. The sigmoid layer of the 'output gate', o_t, decides the cell state information that will be output. Before, applying the sigmoid, the cell state is run through a tanh layer that helps distribute the gradient values between -1 and 1 and prevent the vanishing/exploding gradient problem.

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{2-8}$$

The LSTM cell learns this required temporal dependency by back propagating it's gradient through a time horizon during the network training. For the proposed network, the value of this horizon is set to 10 while training. This means that the network makes use of a sequence of 10 consecutive laser scan readings during the training and learns how these scan readings change over time while a mobile robot navigates.

The LSTM cell in the Laser Scan Channel has 256 units. The number of these units is higher than the size of the input encoding to the cell, i.e. 128. The number of units in the LSTM cell is kept higher as the purpose of the cell is to learn the temporal dependency between the laser scan readings and not to further encode them. A higher number of units ensures that none of the input feature dominates the other and all the input features are equally taken into account while learning the temporal behaviour.

2-1-2 Goal Information Channel

The second sub input channel deals with the goal information provided to the network. This goal information is available in the form of normalized relative polar coordinates in a 2D plane, with the first coordinate containing the relative distance between the current robot and goal positions and the second polar coordinate containing the angle between the straight line joining the robot's and goal's position and the current orientation of the robot. If the robot is currently at (x_r, y_r) with an orientation of α in a 2D Cartesian plane and it's aim is to reach the goal position, (x_g, y_g), the required normalized relative polar coordinates, $goal_{info}$ can be derived using following set of equations:

$$\begin{aligned} d &= \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2} \\ d &= 2(1 - \frac{min(d, d_m)}{d_m}) - 1 \end{aligned} \tag{2-9}$$

$$\begin{aligned} \theta &= \tan^{-1}(\frac{y_g - y_r}{x_g - x_r}) - \alpha \\ \theta &= \begin{cases} \theta + 2\pi & \frac{\theta}{\pi} < -1 \\ \theta - 2\pi & \frac{\theta}{\pi} > 1 \end{cases} \end{aligned} \tag{2-10}$$

$$goal_{info} = [d, \frac{\theta}{\pi}] \tag{2-11}$$

Both polar coordinates are normalized in the interval, $[-1, 1]$ and the maximum possible distance to a goal position from the robot, d_m is set to 20 meters. Similar to the Laser Scan Channel, the Goal Information Channel also consists of a LSTM cell with 32 units and it directly receives the relative polar coordinates of the robot as input. The LSTM cell in this sub channel focuses on capturing the change in the relative angle between the robot and the goal as the robot approaches the goal position and it is observed that the output angular velocity of the mobile robot is highly influenced by this angular data information.

2-2 The Output Channel

In the first half of the proposed network, useful features are extracted/encoded from different inputs to the network and the temporal relation between the consecutive readings of these inputs is also learnt by their respective sub channels in the said network. The next step for this network is to combine these different sets of information readings and learn how they collectively affect the robot navigation behaviour over time.
In order to achieve this, the information output by each of the LSTM blocks in the first half of the network is concatenated with one another and sent as an input vector to another LSTM block. This third LSTM block consists of 288 units and learns the collective temporal behaviour of all the inputs.
Having captured the temporal behaviour of the robot's local environment, the last step for the network is to use this behaviour and output necessary control commands in the form of linear and angular velocities that enable the mobile robot to navigate safely to it's destination. The output of the third LSTM block is fed through two fully connected layers, the first of which has 173 units and the second one having 60 units. The result from these fully connected layers

is input to the output layer consisting of 2 units, 1 unit each corresponding to the linear and angular velocity.
Finally, the linear and angular velocity values from the output layer are fed through the Sigmoid and Tanh activation functions in order to limit their numerical values between 0 and 1 and -1 and 1 respectively.

2-3 Loss Function for Supervised Learning

While training with Supervised Learning approach, the network is optimized based on $|F_\theta(y, g) - u_{expert}|$, the difference between predictions of the network and the expert planner. Hence, there are two possible candidate loss functions, Mean Squared Error (MSE) and Mean Absolute Error (MAE). MSE is quite sensitive to large errors/outliers. It is generally used for applications in which large errors causes way more harm than smaller errors. On the other hand, MAE treats both the large and small errors equally. For the End-to-End planning, both small and large errors are required to be treated equally as even a slight change in the output control commands can turn out to be a difference between robot successfully completing a trajectory or robot colliding with an obstacle. Hence the loss function for the Supervised Learning approach is given by the following MAE function:

$$J_k = \frac{1}{N_B} \sum_{j=i}^{i+N_B} \frac{1}{T} \sum_{t=1}^{T} |F_{\theta_k}(y_{j,t}, g_{j,t}) - u_{exp.,j,t}| \tag{2-12}$$

Here, k is the k^{th} iteration, N_B is the batch size and T is the length of horizon for back propagating the gradient through time. The loss function makes use of Adam Optimizer for updating network parameters during training.

Chapter 3

Data Collection

Having proposed the End-to-End model that can enable a mobile robot to navigate autonomously in it's environment, the very next step is to collect a suitable training data in order to train the model for the required task using Supervised Learning. This chapter discusses the simulation environment, the expert planner and data types used to obtain the required data.

3-1 Simulator

There are two ways to collect a dataset required to train a Deep Neural Network. Either one can directly collect the data from the real world or one can obtain a close enough simulation of the real world and collect the data from it instead. But collecting data on a real robot in a real world environment can be quite expensive and time consuming. As a result, a simulation for a differential drive mobile robot is prepared in order to collect the required data in comparatively time efficient and inexpensive manner. This section focuses on the simulator considered for collecting the required data.

3-1-1 Gazebo

Gazebo[1] is an open source 3D robot simulation with the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. It also has the ability to simulate different types of sensors like Light Detection and Ranging (LiDAR) and RGB-D cameras and collect the desired navigation data. At heart, Gazebo is a robust physics engine that takes various physical factors into account while simulating the robot in an environment. This property plays a significant role in bridging the gap between real and simulated world while testing the resulting algorithms. It is also possible to visualize the sensor data collected in Gazebo on Rviz. All these factors make Gazebo an ideal candidate to simulate a differential drive robot in a static environment (see Figure 3-1).

[1] http://www.gazebosim.org/

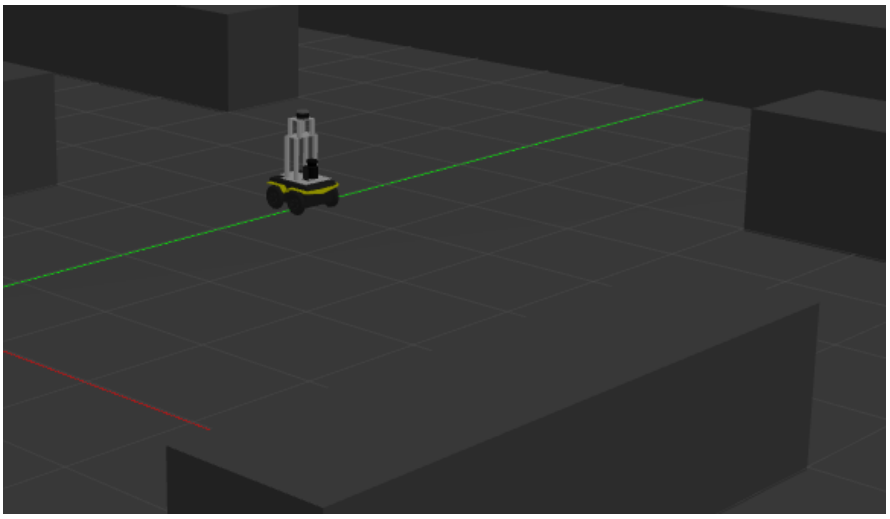


Figure 3-1: Gazebo simulation of Jackal differential drive robot

3-2 Expert Planner

After setting up the simulation environment, the next step is to select an appropriate method to make the robot navigate in that environment. One of the ways to do that is to manually control the robot in the simulation using an external device like a joystick or a keyboard. But such an approach has some major drawbacks as the quality of the recorded data or the robot motion is directly affected by the driving skills of the person controlling the simulated robot. This issue can be resolved by using an expert motion planner that can make the robot navigate autonomously to it's goal position in a safe and efficient manner. An expert motion planner is quite advantageous in the fact that it takes into account the the robot model and state while making a decision and it's operation is heavily influenced by the cost function of the task at hand. This section describes the motion planner used to collect the dataset.

3-2-1 ROS Move Base

Move Base is a Robot Operating System (ROS) package, that given a goal location in the world, tries to reach it with a mobile base. The package links together a Global and a Local planner to achieve the required navigation task. For each of the planners, a costmap is also maintained in order to achieve the desired goals. The data is collected in static environments with Dijkstra's algorithm being used as the Global planner and Dynamic Window Approach (DWA) being used as the Local planner. The data collection is carried out on 4 different 20m x 20m closed maps with obstacles placed at random locations (see Figure 3-2). Around 500

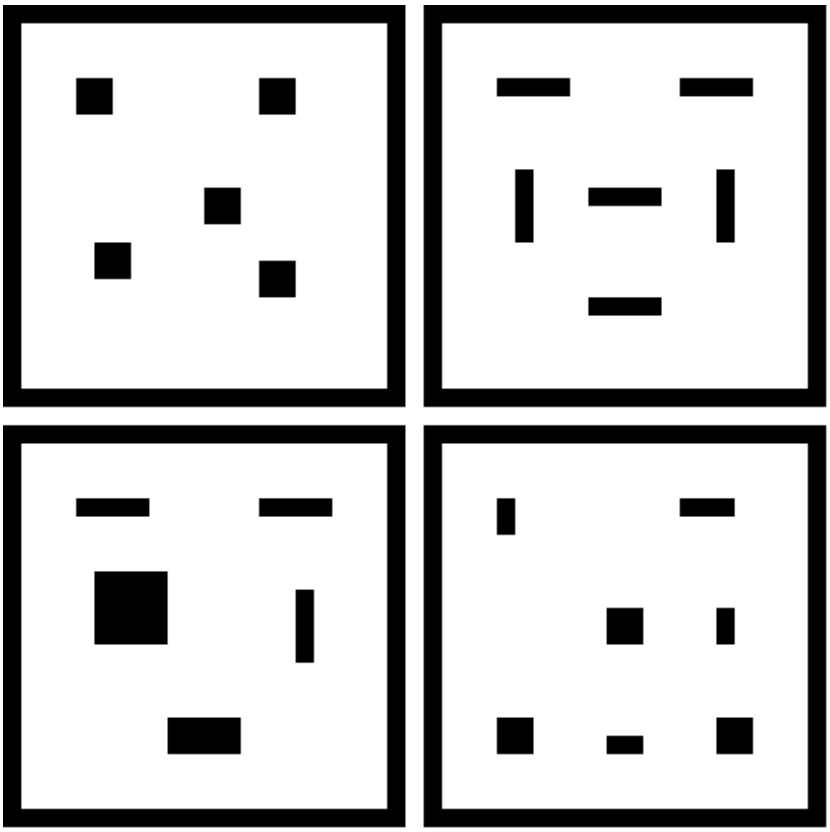


Figure 3-2: Maps used for collecting training data

trajectories are recorded using this planner on each of the 4 maps with each trajectory having a different combination of robot start and goal position. An execution of a robot trajectory on one of the training maps is depicted in Figure 3-3

3-3 Data Recording

Each point in a trajectory using the above mentioned expert planner is recorded at a frequency of 5 Hz and consists of the following data keys in the form of a python dictionary [2]:

- **Laser Scan**: This consists of vector of 2D laser scan readings obtained from the simulated LiDAR having a maximum range of 30 m. The LiDAR collects a set of 720 readings per revolution over a 360° field of view. This vector is used as an input to the Laser Scan channel in the network proposed in the previous chapter.
- **Binary Occupancy Grid**: This field stores the Cartesian representation of the recorded Laser Scan readings. The laser scan readings are converted into 60x60 binary occupancy grid. The grid is centred around the robot's local frame and helps provide a 2D visualization of the laser data (see Figure 3-4). The required conversion is carried out with

[2]https://docs.python.org/2.7/tutorial/datastructures.html#dictionaries

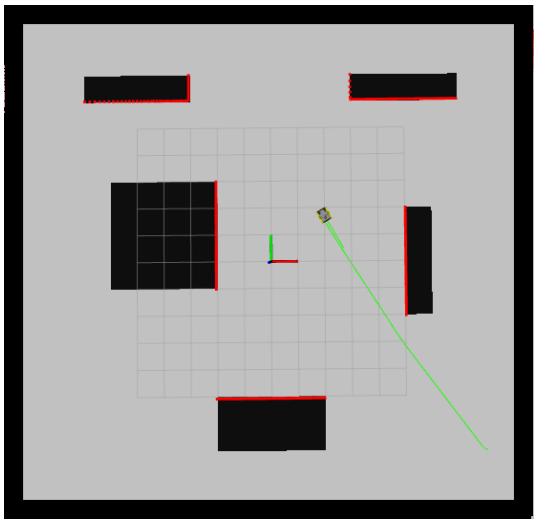


Figure 3-3: Robot Navigating in 20mx20m map using move base. The green line represents the planned path for the robot

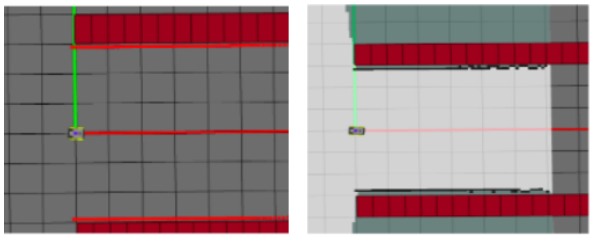


Figure 3-4: Left: Red lines representing the laser scan readings around the robot. Right: Laser scan readings converted to a binary grid with white depicting empty space and black representing occupied space and the robot at the center of the grid

the use of open source ROS package 'Large Maps Framework'[3].

- **Robot Pose**: This consists of an array of robot position values at a given time. The array contains the robot's position in the world frame in x and y direction (in meters) and it's orientation in z direction (in radians).
- **Control Command**: The Control Command contains the linear and angular velocity prediction by the expert planner.
- **Goal Information**: The Goal Information consists of the difference between the desired goal location and the current robot position in Cartesian coordinates in world frame. This data along with the data stored in the 'Robot Pose' key is used to calculate the input for the 'Goal Information' channel of the proposed network.

[3]https://github.com/lama-imr/lama_utilities

Chapter 4

Results and Discussions: Supervised End-to-End Learning

The previous two chapters dealt with the structure of the proposed End-to-End navigation framework and the data collection procedure required to train the said framework. This chapter focuses on the supervised training and testing of the proposed approach and it's analysis compared with other methodologies.

4-1 Network Training

The network is trained on an Intel i7 processor using the data collected by the expert planner as mentioned in the previous chapter. The training dataset consists of 4,000 trajectories comprising of around 2 million data points. A total of $50,000$ iterations are executed with a learning rate of 0.0001 and batch size of 64. Each batch in an iteration consists of 10 consecutive data points from a particular trajectory as the length of the horizon for back propagation through time is set to 10 for network training. As a result, 640 data points from 64 different trajectories are utilized in a single iteration. Each data point consists of a tuple of laser scan data, goal information and the velocity commands of the expert planner.

4-2 Network Testing

The trained network is tested on three maps, completely different from the ones used to collect the training data. Each of the test maps are of the dimensions $20x20$ meters and consist of rectangular and square shaped obstacles. On each of the maps, the simulated mobile robot is made to execute hundred random trajectories. For each current trajectory, the starting point is the goal point of the previous trajectory. If during a trajectory, the robot collides with an obstacle, the robot simulation re-initializes at start point $(0, 4)$. The achieved results are

depicted in Figure 4-1, Figure 4-2 and Figure 4-3 and their statistics are detailed in Table 4-1, Table 4-2 and Table 4-3 respectively.

The maximum distance between the start and goal position for a trajectory is restricted to 20 meters and the maximum time limit for the robot to complete a given trajectory is set to 200 seconds. It is observed that the success rate of the network is around 80% for all three test maps, i.e. the simulated robot is able to successfully reach it's goal 80 out of 100 times. Because of the temporal dependency introduced by the LSTM blocks in the network, it is observed that the resulting trajectories are stable and the robot does not exhibit any erratic behaviour during it's navigation.

4-3 Drawbacks of the Supervised Learning Approach

Although the robot is able to complete it's trajectory using the End-to-End planner 80% of the time, it is observed that it also faces timeouts and collisions for remaining 20% cases. There are mainly two reasons behind these collisions and timeouts, the inertia effect of the LSTMs and the deterministic nature of the network. The effects of the former are usually observable during longer trajectories while the latter is more prominent in shorter trajectories.

4-3-1 The Inertia Effect of the LSTMs

Besides providing the much needed temporal dependency that results in a stable execution of trajectories, the use of LSTMs in the networks also comes with some pitfalls. Since, the current network output depends on the previous output control commands, the LSTM cells end up introducing some inertia in the network that prevents any drastic changes to the output velocities when the robot might actually require them. This can lead to the robot colliding with the obstacles or not completing it's trajectory within the stipulated time period. There are usually two particular behaviours that the robot exhibits due to the said inertia effect:

- If the goal position is at a large distance from the robot's starting position and there is a significant amount of empty space available, the robot tends to moves with a larger linear velocity towards the goal in order to complete the trajectory within a given time frame. But if an obstacle suddenly appears in the robot's path, the chances of the robot colliding with that particular obstacle increase because of the inertia induced by the LSTMs. The network prevents the robot to drastically alter both the linear and angular velocity because the robot has been moving with a certain velocity in the empty space for a considerable amount of time. The network does introduce some change to the velocities in order to avoid a collision but these changes can sometimes be just not enough to safely avoid the obstacle.

- While executing a long trajectory, as the robot comes closer to the goal, there might be an instance when the robot's heading doesn't perfectly align with the goal and as a result the robot is required to significantly alter it's orientation in order to reach the destination. To make such a change possible, the magnitudes of robot's linear and angular velocities need to reduce and increase respectively. But if the robot has been moving with a higher velocity, the network isn't able to significantly alter the velocity due to the LSTM induced inertia. As a result, instead of taking a turn towards the goal, it drifts and gradually changes it's velocity while it continues to move around the goal in circles which occasionally leads to timeouts. This behaviour can be observed in the circular regions formed by robot navigation in Figure 4-1 and Figure 4-3.

4-3-2 The Deterministic Nature of the Trained Network

The supervised learning approach used to train the network is deterministic in nature. As a result, the robot will always have access to one velocity value at a time step rather than a distribution of potential velocity values. This can sometimes turn out to be detrimental for the robot, especially in cases when the goal is nearby and located directly at the opposite end of the obstacle in front of the robot. Since the network learns to mimic the behaviour of the expert planner, it will always try to output the velocity that will take the robot to the goal in the shortest possible time. Hence, the network can discard the velocities that make the robot navigate around the obstacle if the goal is at the opposite side. This leads to three behavioural observations:

- the robot will just stop in order to avoid collision and not move as the view of the goal gets blocked by the obstacle and the deterministic nature of the network prevents the robot to find an empty space in front of it to overcome this blockade, which in turn leads to a trajectory timeout.
- Due to the inertia from the LSTMs and the network's behaviour to make the robot reach it's destination in the shortest possible time, the robot will directly ram into the obstacle in an effort to reach the goal at the opposite end.
- If the robot is present near one of the corners of the obstacle, it will try to rotate and change it's orientation in order to find some empty space in front of it and navigate around the obstacle. But despite this rotation, the magnitude of the linear velocity is not high enough due to the presence of the obstacle in the vicinity and eventually a timeout occurs. Sometimes, a collision can also occur if the robot tries to rotates when it is quite closer to the obstacle.

Table 4-1: Navigation results of End-to-End Planner on Map 5

Total Trajectories	Collisions	Time-outs	Successes	Mean distance to goal before timeout
100	13	7	80	0.65m

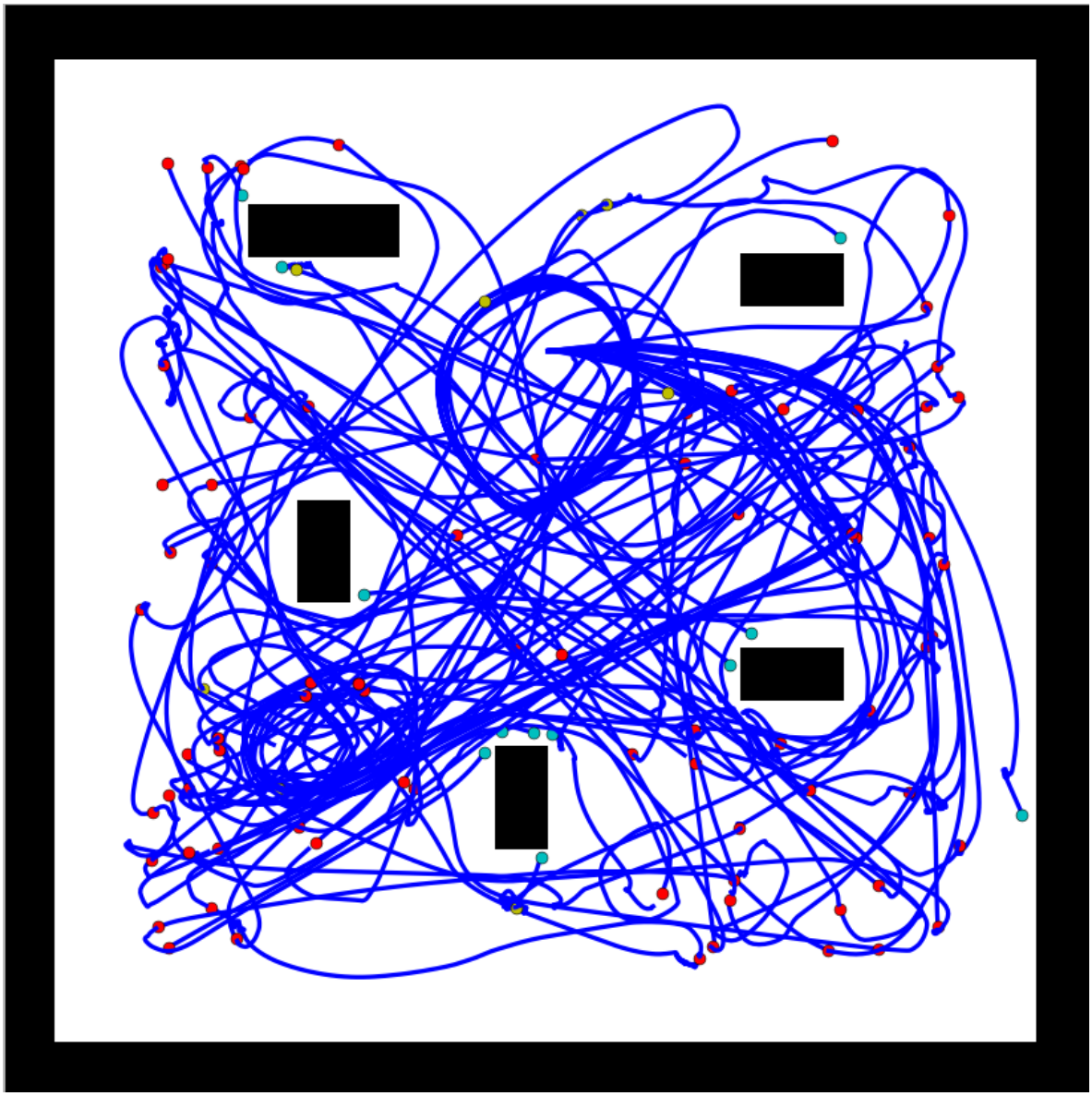


Figure 4-1: Trajectories of the robot using End-to-End Planner with 100 random goals on map no. 5. Red dots represent the goal points of completed trajectories, collisions are represented with cyan and yellow shows the timed out trajectories.

Table 4-2: Navigation results of End-to-End Planner on Map 6

Total Trajectories	Collisions	Time-outs	Successes	Mean distance to goal before timeout
100	12	7	81	0.94m

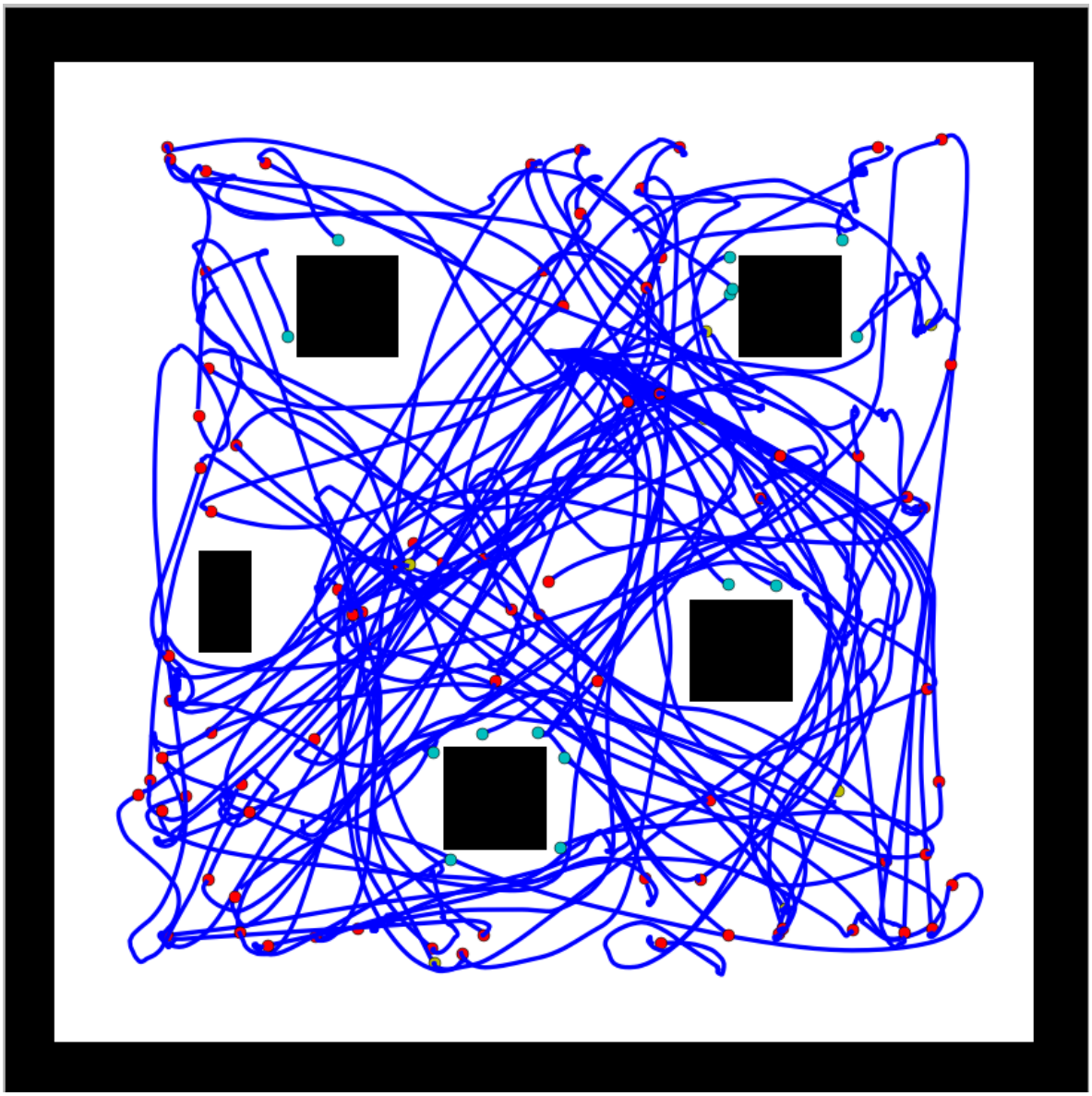


Figure 4-2: Trajectories of the robot using End-to-End Planner with 100 random goals on map no. 6. Red dots represent the goal points of completed trajectories, collisions are represented with cyan and yellow shows the timed out trajectories.

Table 4-3: Navigation results of End-to-End Planner on Map 7

Total Trajectories	Collisions	Time-outs	Successes	Mean distance to goal before timeout
100	12	7	81	0.98m

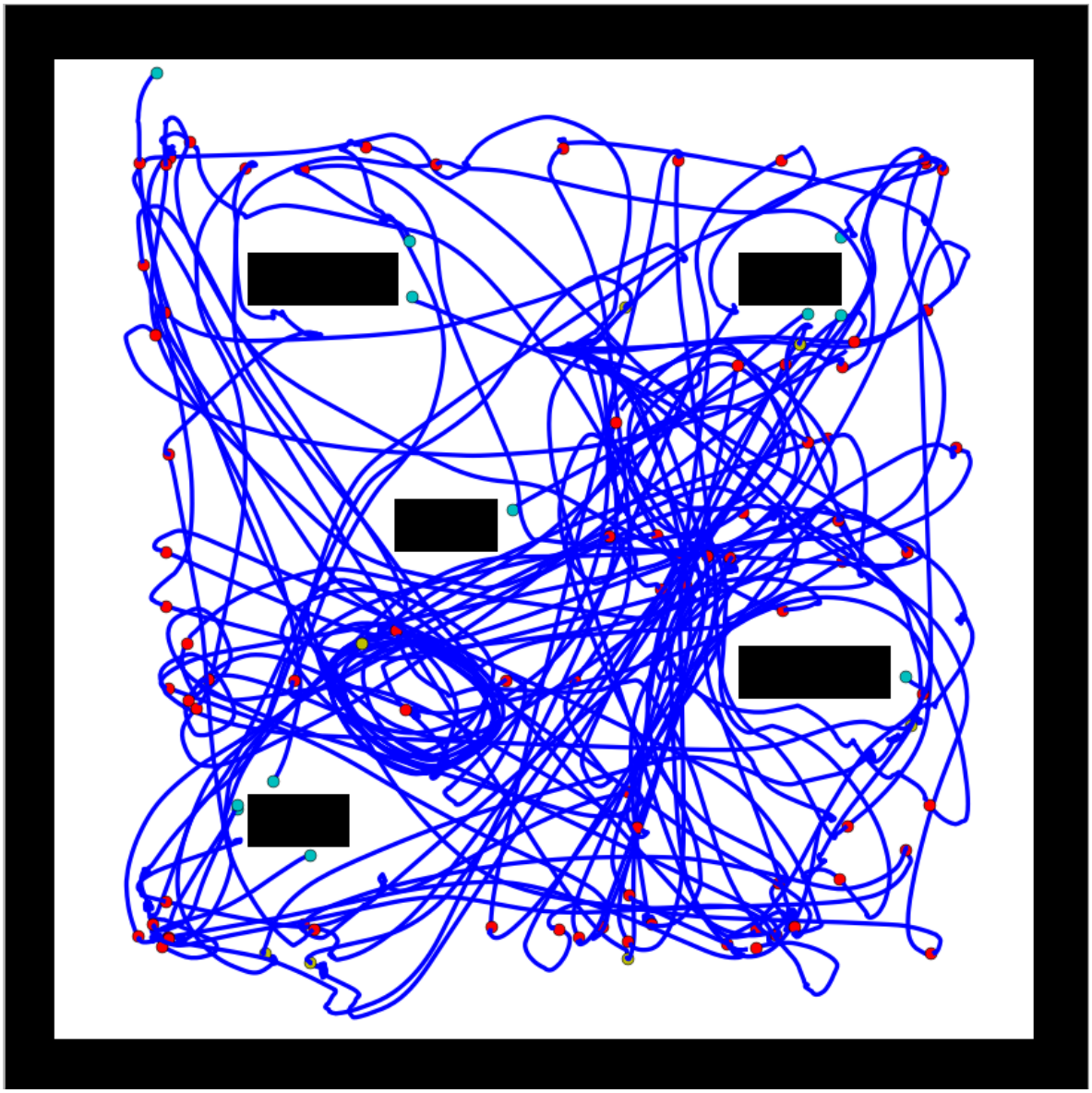


Figure 4-3: Trajectories of the robot using End-to-End Planner with 100 random goals on map no. 7. Red dots represent the goal points of completed trajectories, collisions are represented with cyan and yellow shows the timed out trajectories.

Table 4-4: Statistics for map 5

Planner	Total Distance(m)	Total Time (s)	Mean abs. linear velocity(m/s)	Mean abs. angular Velocity(rad/s)
End-To-End	70.88	227.33	0.3118	0.271
ROS Move Base	54.71	164.8	0.332	0.181

4-4 Comparison with other Methodologies

The performance of the proposed planner is further analyzed by comparing it with that of the expert planner used in supervised training. Both the planners are made to execute a closed loop trajectory consisting of 6-7 goal points on all the test maps. The recorded trajectories are compared using the following metrics:

- Total distance travelled to complete the trajectory
- Total time taken to complete the trajectory
- Mean absolute linear velocity during the whole duration of the trajectory
- Mean absolute angular velocity during the whole duration of the trajectory

The obtained results are visualized in Figure 4-4, Figure 4-5 and Figure 4-6 and their statistics are detailed in Table 4-4, Table 4-5 and Table 4-6 respectively.

4-4-1 Supervised End-to-End Learning vs ROS Move Base

Compared to the Move Base, the Supervised End-to-End planner can sometimes overshoot it's goal because of the LSTM induced inertia and later recover to return to the said goal position. This effect is frequently visible in the closed loop trajectory in Figure 4-4. While the Move Base is comfortably able to avoid obstacles, it is observed that the Supervised End-to-End planner can sometimes take it's time to navigate around a corner of an obstacle by rotating around it's axis in order to find a suitable driving direction. The rotation usually happens if the goal position is near that particular corner of the obstacle as during that time, the robot being both close to the obstacle and the goal has to reduce it's linear velocity and drastically alter it's angular velocity in order to perfectly align itself with the goal (observe point number 4 in Figure 4-4 and point number 6 in Figure 4-6). By comparing the various statistics presented for all three closed loop trajectories, it is observed that the data obtained for the Supervised End-to-End planner is quite comparable with that of the Move Base. Considering the fact that the Move Base has access to both the local and the global information of the navigation environment and the End-to-End planner has access to only it's local information and has no global information available, it can be stated that the End-to-End planner is able to capture the behaviour of it's expert to a significant extent.

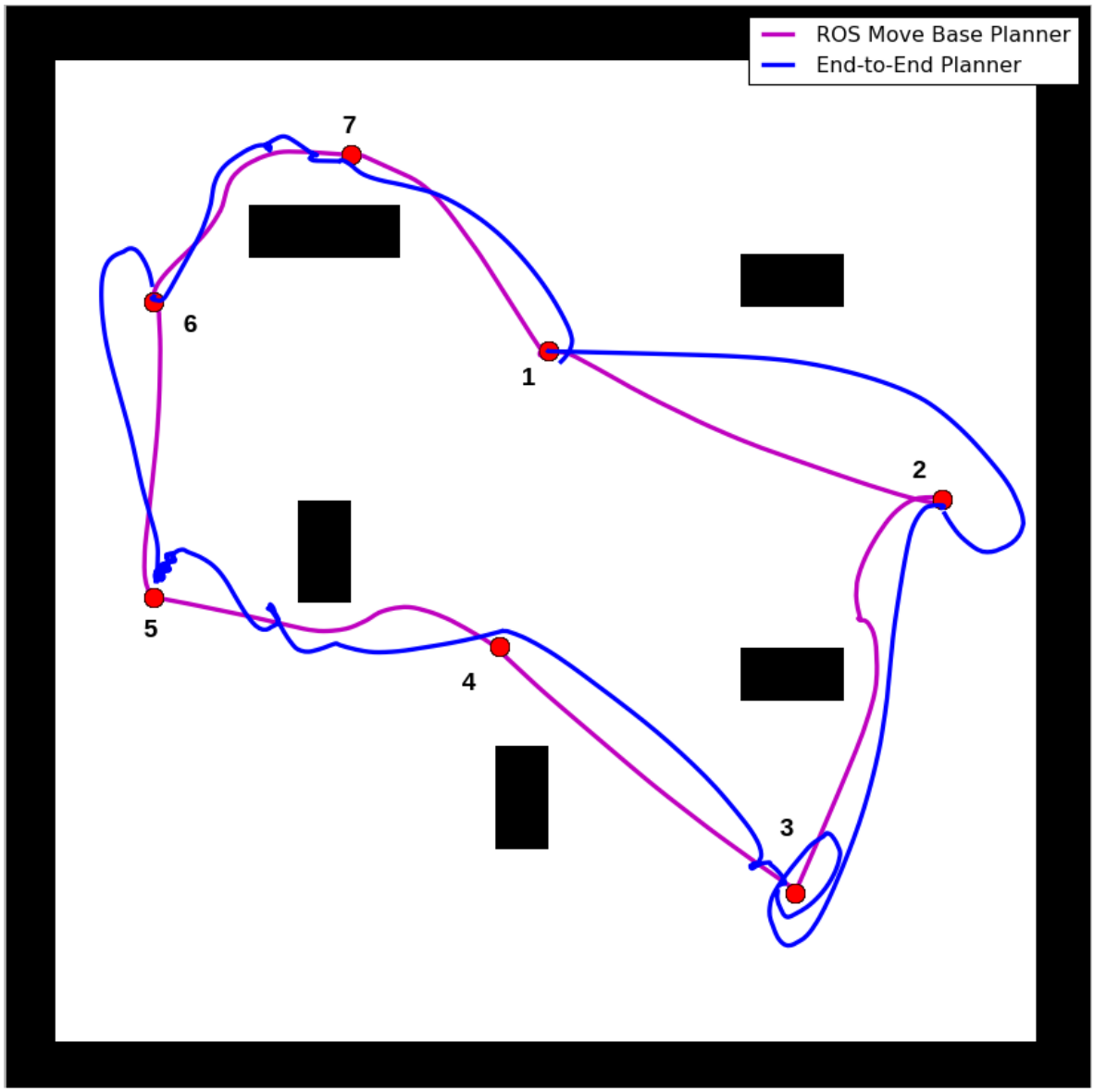


Figure 4-4: A closed loop trajectory executed by two planners on map 5

Table 4-5: Statistics for map 6

Planner	Total Distance(m)	Total Time (s)	Mean abs. linear velocity(m/s)	Mean abs. angular Velocity(rad/s)
End-To-End	47.03	142.7	0.323	0.188
ROS Move Base	47.35	125.3	0.378	0.155

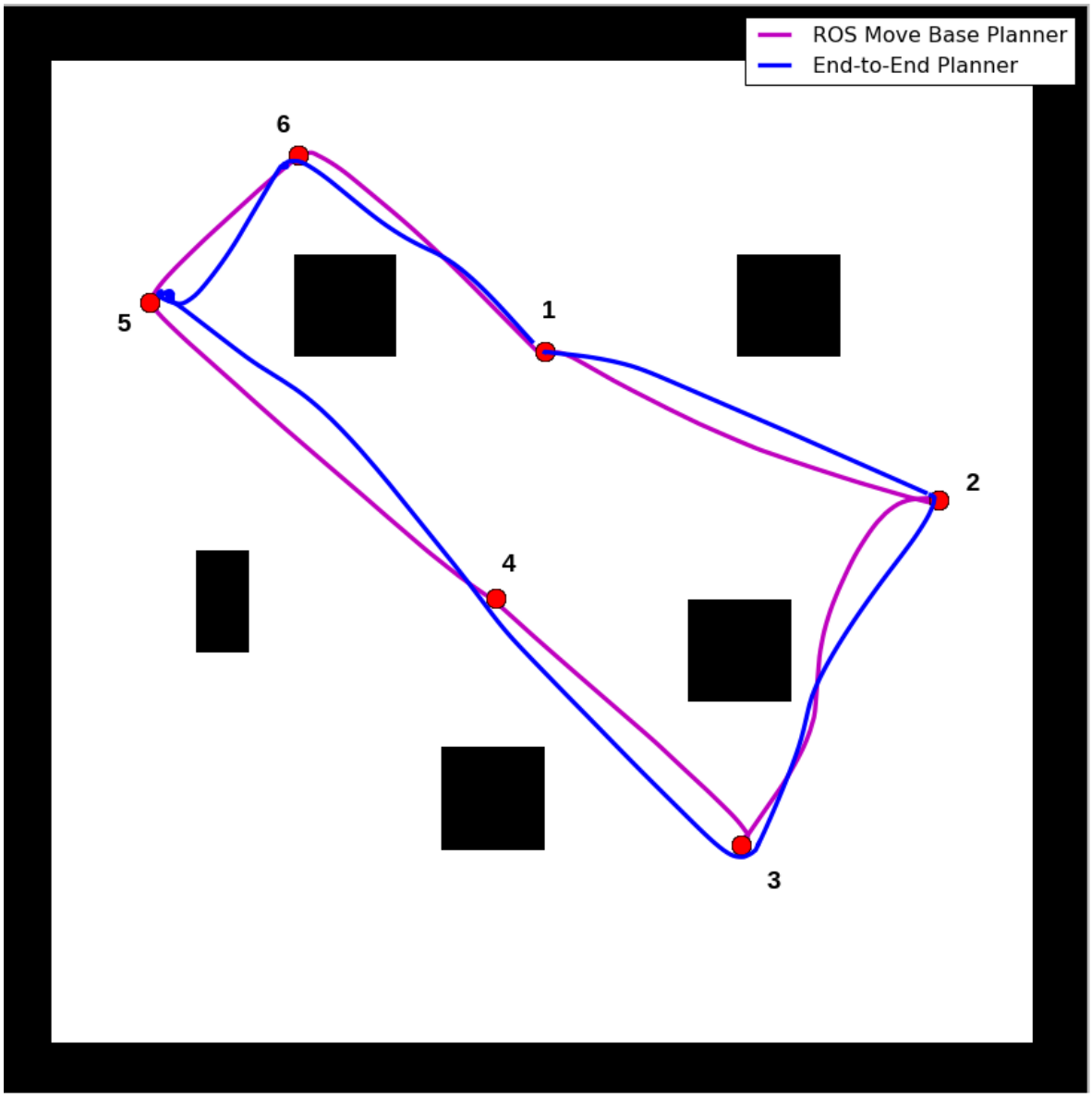


Figure 4-5: A closed loop trajectory executed by two planners on map 6

Table 4-6: Statistics for map 7

Planner	Total Distance(m)	Total Time (s)	Mean abs. linear velocity(m/s)	Mean abs. angular Velocity(rad/s)
End-To-End	53.06	144.65	0.366	0.238
ROS Move Base	52.5	137.6	0.382	0.193

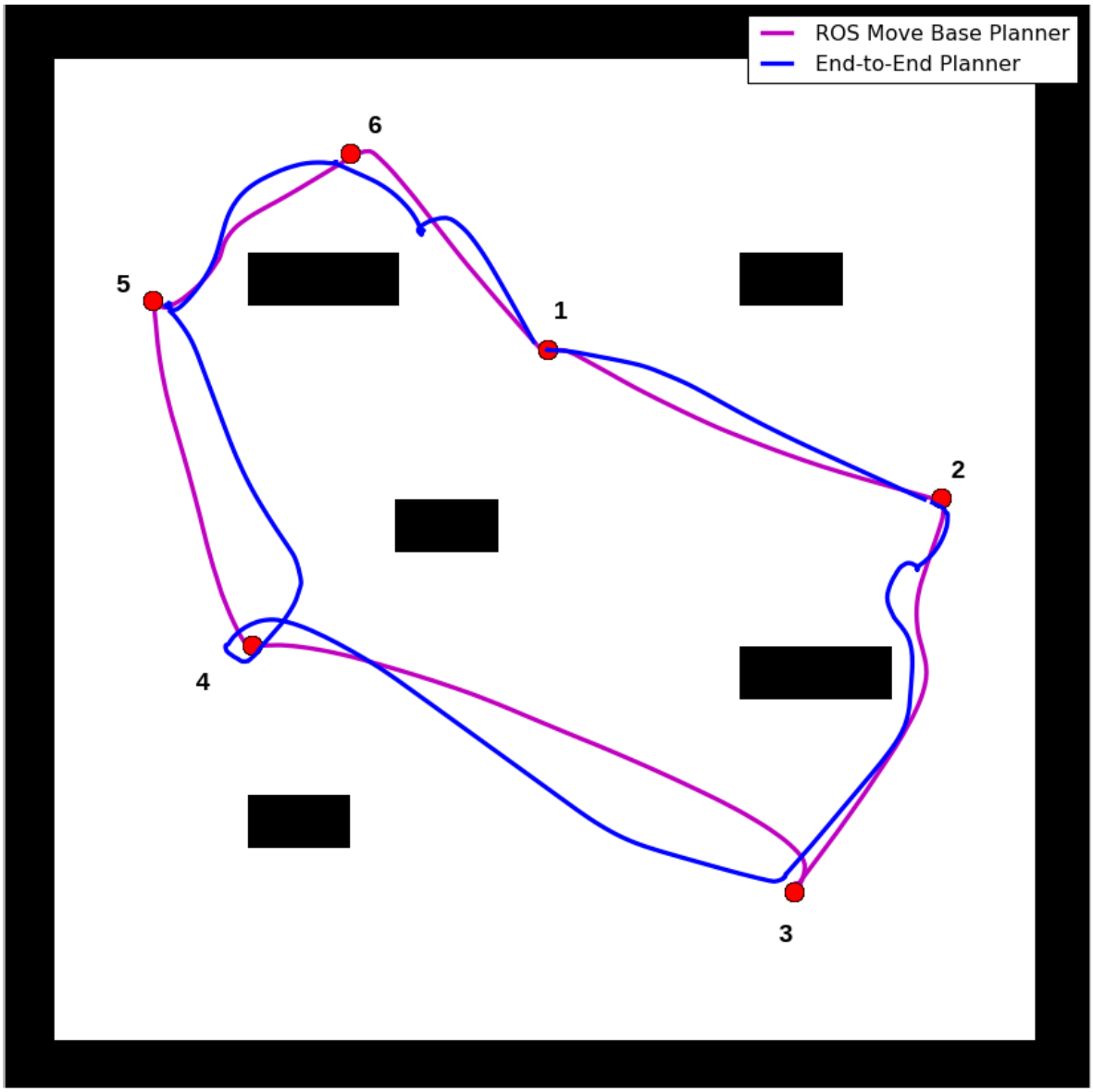


Figure 4-6: A closed loop trajectory executed by two planners on map 7

4-5 Effects of Removing LSTMs from the Network

The supervised end-to-end planner is also compared with networks of similar architecture without having access to all of the LSTM cells in order to further understand the impact, these cells have on the performance of the proposed planner. For the remainder of the section, the planner architecture presented in Figure 2-1 will be referred to as the base architecture. The following four alterations of base architecture are considered to perform the required analysis:

- **No LSTMs**: All the LSTM cells from the base architecture are removed. So, the final network only consists of convolution and fully connected layers.
- **No LSTM (Goal)**: Only the LSTM cell present in the goal information channel is removed.
- **No LSTM (Laser)**: Only the LSTM cell from the Laser Scan channel is removed.
- **No LSTM (Concat.)**: Only the LSTM cell concatenating the information from the input channels is excluded.

All the networks barring the one with no LSTMS at all, are trained with the same specifications as the ones presented in 4-1. For the 'No LSTMs' network, there is no back propagation through time and the whole training data is randomly shuffled to avoid temporal dependency between any two consecutive input data points to the network. The networks are tested with 100 trajectories on each of the three test maps and are also made to execute a closed loop trajectory on one test map. The results are depicted in Table 4-7 and Figure 4-7 and are discussed in the following subsections.

4-5-1 No LSTMs

This version of the network has no LSTM cells present. As a result, the network lacks the ability to learn the change in the behaviour of the robot in it's local environment with time as it approaches the goal. This lack of ability to capture the temporal changes is visible in the trajectory results of Figure 4-7. Based on the initial laser scan and goal information data, the planner is able to initiate the robot motion in the direction of the goal region, but this motion continues to remain in a straight line for majority of the time and the planner fails to make the required angular adjustments as the trajectory progresses. As a result of this, the planner fails to reach 5 out of 6 way points with an average of 4.07 meters of distance to a way point before time out and has a mean absolute angular velocity of only 0.055 rad/sec throughout the execution of the said trajectory.

4-5-2 No LSTM (Goal)

Since the goal information channel is without a LSTM cell, there is no temporal relation established between two consecutive inputs to the said channel. The effects of the lack of the temporal relation are clearly observable as the robot comes closer to it's goal. In the vicinity

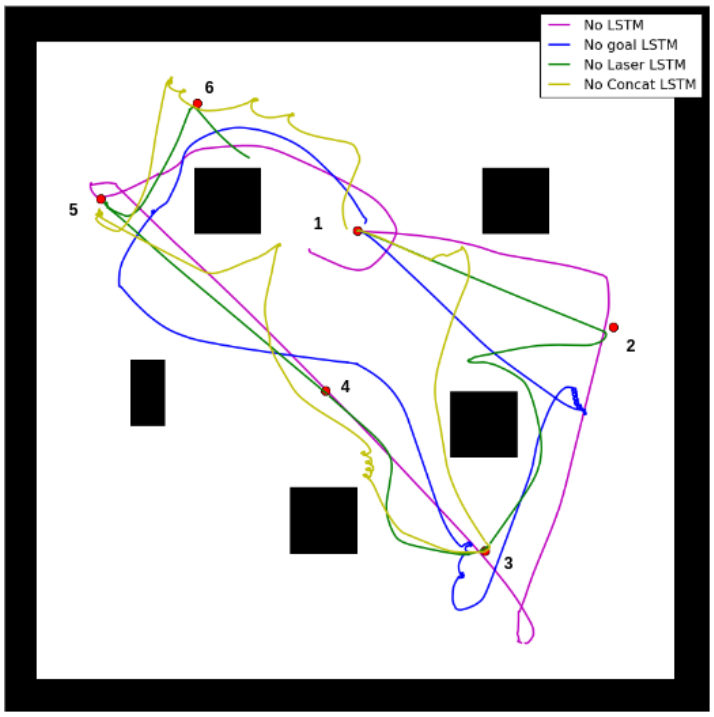


Figure 4-7: Attempts of closed loop trajectory by different planners without LSTM on map 6

of the goal, the magnitude of the linear velocity reduces and the angular velocity which is heavily influenced by the relative angle between the robot and it's goal is required to undergo changes in order to bring the said relative angle to zero. But since, the network does not have the ability to establish the relation between two consecutive goal information readings, the resulting transitions in the angular velocity are often unstable and the robot fails to reach the goal frequently. In the trajectory execution in Figure 4-7, the robot fails to reach 5 out of 6 way points with an average of 1.911 meters distance before a timeout.

4-5-3 No LSTM (Laser)

In this case, the network lacks the ability to establish the temporal relation between two consecutive laser scan readings. As a result, the network is no longer able to anticipate it's future local environment based on it's current control action. This leaves the network prone to executing unstable control commands, examples of which are visible in Figure 4-7 when around point 2, the robot suddenly deviates from it's path and directly rams into an obstacle after reaching point 6.

4-5-4 No LSTM (Concat.)

In this particular case, even though the network is able to establish the temporal relation for each individual input channel, the absence of the LSTM cell responsible for concatenating all the information from input channels makes it difficult for the network to learn how the different sensor inputs influence each other with time in order to achieve the desired navigation

Table 4-7: Statistical comparison of models withh no LSTMs to the base planner

Planner	Collision %	Time Out %	Success %
Base Planner	12.33	7.00	80.67
No LSTMs	41.33	36.33	22.33
No LSTM (goal)	20.67	45.33	34.00
No LSTM (laser)	39.67	17.00	43.33
No LSTM (concat.)	23.00	28.00	49.00

behaviour. As a result in Figure 4-7, the trajectory executed by this particular planner frequently consists of the execution of unstable control commands.

4-6 Key Takeaways

Having compared the performance of the proposed network with other methodologies, it can be concluded that the proposed Supervised Learning approach is able to mimic the behaviour of the expert planner to a significant extent. However, this Supervised Learning method is not perfect. Though the method is able to produce stable trajectories for the robot, it is not fully collision proof. As observed, the Supervised Learning based planner is sometimes not able to avoid collisions, it may either be because of the inertia from the LSTMs or because of the deterministic nature of the adopted approach. This vulnerability of the Supervised Learning approach also stems from the fact that during training, the network doesn't have any access to collision data and there is no way to penalize the network for attempting to take the robot to unsafe collision states. Based on the results and discussions observed so far, the first two of three research questions presented in the beginning can be addressed as follows:

- **Can an End-to-End motion planner learn to navigate to a desired goal position by just using the raw sensor reading from the robot's local environment?**
  It is observed that the proposed End-to-End motion planner, by making use of the raw laser scan readings and relative goal information from its local environment, is able to complete a given trajectory in majority cases. Hence, it can be concluded that an End-to-End planner can learn to navigate by just using raw sensor data from it's local environment.
- **Is a Supervised Learning based approach sufficient enough to capture the desired navigation behaviour?**
  The Supervised Learning based End-to-End planner is able to capture the expert's navigation behaviour to a significant extent but still this method alone is not sufficient enough as it struggles to avoid collisions in some cases. A possible solution is to combine this approach with another learning technique like Reinforcement Learning and learn a new policy that is better equipped at avoiding obstacles.

Chapter 5

Supervised-Reinforcement Learning

In the previous chapter, it was concluded that the Supervised Learning based end-to-end planner is not fully collision proof. One of the ways to improve it's performance is to further train it with a different Machine Learning approach like Reinforcement Learning, where the network can have access to the collision data and can be suitably penalized for entering unsafe states in an environment. This chapter discusses the structure of the proposed Reinforcement Learning approach along with the reward function and the optimization algorithm used to learn the required policy. The approach discussed in this chapter is based on the work done on Reinforced Imitation by [15].

5-1 Reinforcement Learning Approach

For a Markov Decision Process, $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(\cdot|s_t, a_t) : \mathcal{S} \times \mathcal{S} \times \mathcal{A} :\rightarrow \mathbb{R}_+$ is the transition probability distribution, $\mathcal{R}(\cdot, \cdot) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function and $\gamma \in [0, 1]$ is the discount factor, Reinforcement Learning (RL) aims to find a policy π_θ, mapping states to actions and parameterized by θ, that maximizes the expected sum of discounted rewards,

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t \cdot r(s_t, \pi_\theta(s_t))] \tag{5-1}$$

Here, T is the time horizon of a navigation trajectory. In this work, s_t is represented by the laser scan readings and the goal information and a_t is represented by the control commands.

The RL based approach is formulated using Actor-Critic methodology, where the Actor learns the desired policy $\pi(a|s)$ and the Critic learns to approximate the value function. The Actor then makes the changes to it's parameters based on the update direction suggested by the Critic. In the proposed algorithm, the Actor network is the same as the one in Figure 2-1. But in this case, the Actor learns a stochastic policy in the form of a 2D Gaussian distribution with the network outputs being the mean values of linear and rotational velocities and a

2D standard deviation which is a separate learn-able parameter. During training the Actor model is initialized with the weights learnt by the Supervised Learning model, thus making the algorithm, a Supervised-Reinforcement Learning method.

The Critic model on the other hand, consists of three Fully Connected hidden layers of size 512, 256 and 128 that take as input the states observed by the robot of size 722, which include the laser scan readings and the goal information in relative polar coordinates. The output of the Critic network is of size one that contains the value function.

5-1-1 Reward Function

The reward function for the RL problem is designed such that the agent learns to navigate to the goal in the shortest possible time while avoiding collisions with obstacles in it's environment.The reward function is formulated as

$$r(s_t) = \begin{cases} 10, & success \\ -(d(s_t) - d(s_{t-1})), & otherwise \end{cases} \tag{5-2}$$

The next step is to assign a value for $d(s)$. If $d(s)$ is set to 0 for all s, only a minimum amount of information gets encoded to carry out the required task. This 'sparse reward' makes the learning process difficult as all the actions taken in an episode get credit for their outcome regardless of whether they contributed to it or not. Another alternative is to use Euclidean distance between s and goal as the value for $d(s)$. This helps provide continuous feedback for each action by rewarding/penalizing the agent for getting closer/further to/from the goal. However, this approach does not consider the placement of obstacles in the environment. In order to overcome this issue, the value of $d(s)$ is set as the distance between s and the goal along the 'shortest feasible path'. This path is computed by making use of the Dijkstra algorithm.

5-1-2 Constrained Policy Optimization

In a standard RL approach, the agent explores by trying many different policies using trial and error. During the exploration period, the agent can exhibit an unsafe behavior in order to achieve the optimum. For mobile robots, that are required to learn a navigation behaviour, this can turn out to be a very serious problem. Hence, the need arises to address a very important concern among the RL algorithms: safety. An autonomous system is considered safe if it's failure modes are rare and happen within some specified rate. This paves way for constrained RL formulation as a means to incorporate safety into RL and ensure that every exploration policy also satisfies the said safety constraint. Constrained Policy Optimization (CPO) helps satisfy these needs.

Local policy search is a standard iterative way to learn policies. Policy gradient methods are local search methods that use modifications of stochastic gradient descent to optimize $J(\theta)$ with respect to policy parameters θ. However, they suffer from high variance in gradients that results in undesirable large updates to the policy. Trust Region Policy Optimization (TRPO) methods [16] are a type of local policy search algorithm that reduce the model variance and

provide stability by ensuring that each new policy is close to the old one in terms of average KL-divergence. CPO [17] is a trust region method for constrained RL which approximately enforces the constraints in every policy update. Using the approximations of the constraints, it predicts the amount by which the constraint costs may change after an update and then chooses the update that will most improve the performance while keeping the constraint costs below the specified limits.

Given a cost function $\mathcal{C} : \mathcal{S} \times \mathcal{A} :\to \mathbb{R}$, let $J_C(\theta)$ indicate the expected discounted return of π_θ with respect to this cost

$$J_C(\theta) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t \cdot C(s_t, \pi_\theta(s_t))] \tag{5-3}$$

CPO returns a solution to the following problem

$$\theta^* = \arg \quad \max J(\theta), \quad \text{s.t.} \quad J_C(\theta) \leq \alpha \tag{5-4}$$

In the proposed RL approach, the collision avoidance is encoded through a constraint on the expected number of crashes allowed per episode. Considering $\mathcal{S}_c \subset \mathcal{S}$ as a set of collision states, the required state dependent cost function is defined as

$$c(s_t) = \mathcal{I}(s_t \in \mathcal{S}_c) \tag{5-5}$$

Here, $\mathcal{I}$ is the indicator function. For the proposed model, the value of α is set to 0.4 as per the empirical results derived by [15].

Chapter 6

Results and Discussions: Supervised-Reinforcement Learning

The previous chapter dealt with the details of the proposed Reinforcement Learning based method. This chapter focuses on the proposed method's training and testing and it's analysis compared with Supervised Learning based approach.

6-1 Network Training

In order to train the RL based policy, a considerably large amount of data is required. The training is carried online and hence a real-time implementation can take weeks to learn the desired navigation policy. In order to address this time issue, a simulator is required that can significantly accelerate the real-time data collection procedure. Stage [18] provides this ability and is thus the simulator used for training the RL policy instead of Gazebo. But since an open source simulation for the Jackal UGV on Stage is not available, another differential drive robot in the form of Turtlebot is considered to collect the required data in the chosen simulation environment.The network training is carried out on an Intel Core i9 CPU.

During training, the Actor part of the network is initialized with weights learnt from Supervised Learning. In order to overcome the drawbacks of the deterministic approach of the Supervised Learning based planner as stated in chapter 4, the Actor is made to learn a stochastic policy where the actions are sampled from a 2D Gaussian distribution, with the standard deviation being a separate learn-able parameter and the network returning mean linear and angular velocities as it's output. During the training, a random start and goal position are selected and the robot experience samples are collected by running an episode using the current policy for a fixed number of time steps or until the robot reaches the goal. The value of these fixed time steps is set to 300 and a batch of samples are collected from multiple episodes for each epoch of training.

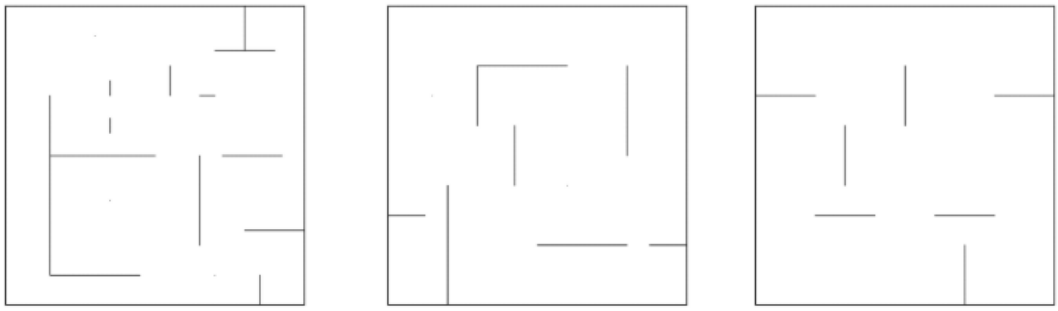


Figure 6-1: Training maps used in Stage simulator to train the Supervised-Reinforcement model

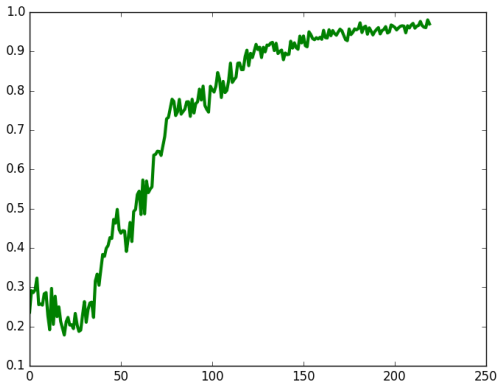


Figure 6-2: Success rates during model training

The network is trained on three different training maps of size 10×10 meters (Figure 6-1). During RL, the training environment is uniformly sampled from the three training maps. Each epoch, for which around $60,000$ time steps are considered, the accelerated Stage takes around $250 - 300$ seconds to gather the required number of experiences from the simulation. Figure 6-2 shows the success rates of the model during RL training. In this curve, a kink is observed during the initial phase of the training. This kink occurs due to the fact that during this phase of the training, the model focuses heavily on learning the collision avoidance behavior, which makes it's approach quite conservative in the beginning, thus leading to frequent time-outs and lower success rates for the different episodes executed in an epoch.

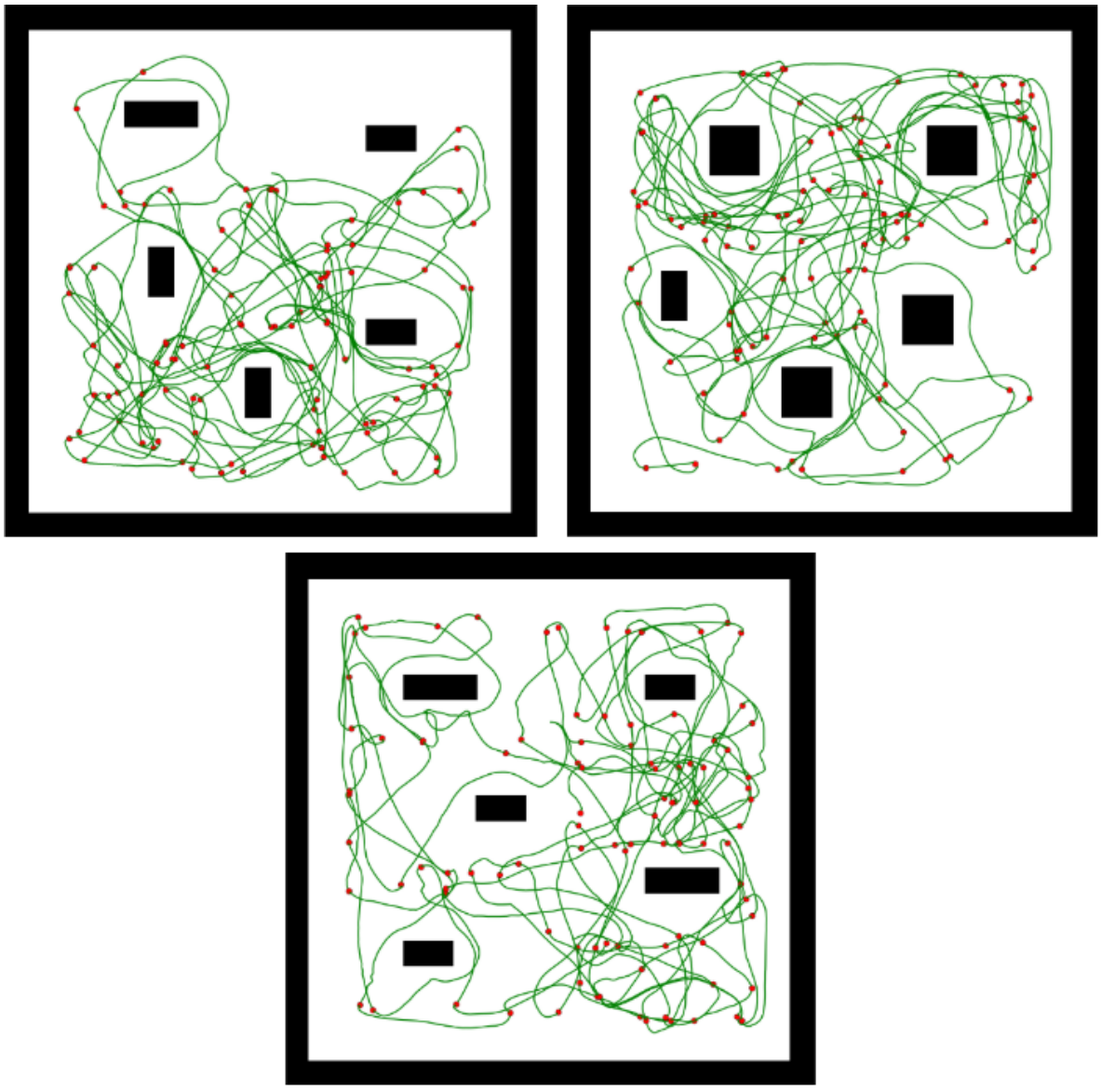


Figure 6-3: Performance of the Supervised Reinforcement Learning based approach on three test maps used for Supervised Learning. 100 Random trajectories are generated on each map with the goal point of one trajectory acting as the start for the next. Successfully reached goals are marked by red dots.

Table 6-1: Navigation results of Supervised-Reinforcement End-to-End Planner on new test map

Total Trajectories	**Collisions**	**Time-outs**	**Successes**	**Success %**
200	0	6	194	97%

6-2 Network Testing

Similar to the Supervised Learning in chapter 4, the trained model is tested on three different test maps with the Jackal simulation on Gazebo. The stipulations for testing the modified network are kept the same as in the previous case. It is observed that for all the three maps, the new network is able to overcome it's previous issues like LSTM induced inertia to execute collision proof trajectories and achieve a success rate of 100% without any collisions and time-outs (see Figure 6-3). But the difficulty level of these test maps is comparatively less than the ones being used to learn the Reinforcement Learning based policy. In order to make sure that the trained model is robust to changing environments and has really improved it's performance, a new test map is created with a difficulty level on par with that of the training maps. The new map also has the dimensions of 20×20 meters. A total of 200 random trajectories are executed on this map with the same conditions as the previous maps and it is observed that every single trajectory is collision proof and the network is able to achieve a success rate of 97% on this particular map (see Figure 6-4 and Table 6-1), thus verifying it's robustness and superior collision avoidance capabilities, compared to the Supervised Learning approach.

6-3 Comparison with Supervised Learning Approach

Supervised Reinforcement Learning (S-RL) approach is able to address the pitfalls suffered by it's Supervised Learning (SL) based counterpart. Based on the results presented in Figure 6-5, the S-RL method takes less amount of distance to complete a closed loop trajectory. It is also a more stable planner with a comparatively smaller magnitude of mean angular velocity over the duration of the trajectory.

It has learnt the ability to avoid overshooting the goal position (observe point 11 in Figure 6-5) and unlike Supervised Learning, it does not struggle with the robot's orientation when navigating around an obstacle's corner. It is also able to keep the robot at a safe distance from the obstacles while the SL based planner would occasionally navigate quite close to the obstacles, a behaviour it inherited from it's expert, ROS Move Base.

The SR-L method has also learnt to counter the negative effects of the LSTM induced inertia. In order to achieve the desired results, it is observed that the SR-L based planner steadily starts decelerating linearly in the second half of a trajectory. As a result of this deceleration, the planner is able to comfortably adjust it's angular motion in order to align itself with the goal and hence avoid situations where it has to drift around the goal in circles or overshoot

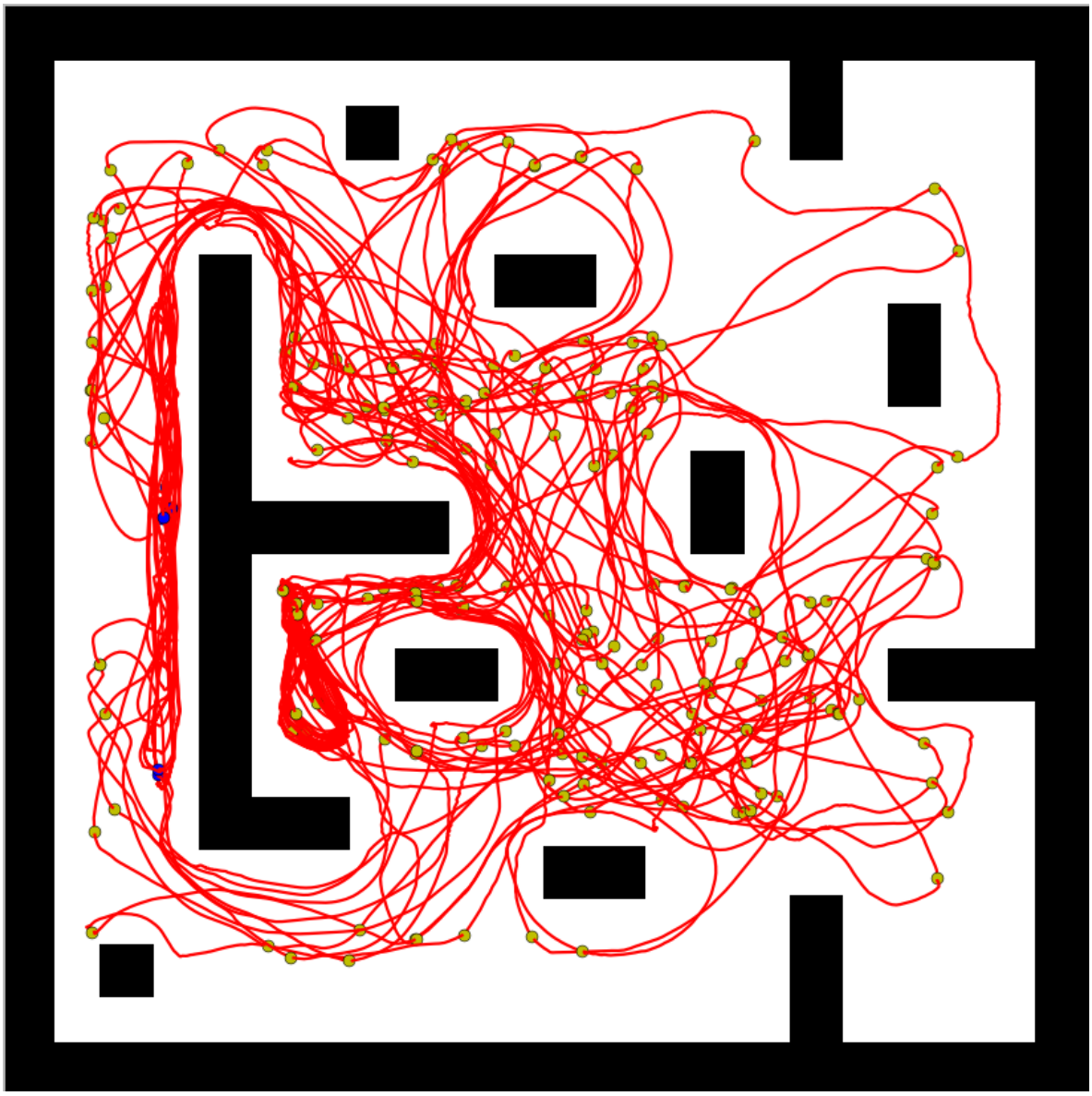


Figure 6-4: Performance of the Supervised Reinforcement Learning based approach on a new test map with more obstacles. Successfully reached goals are depicted by yellow dots, time-outs by blue and crashes by cyan.

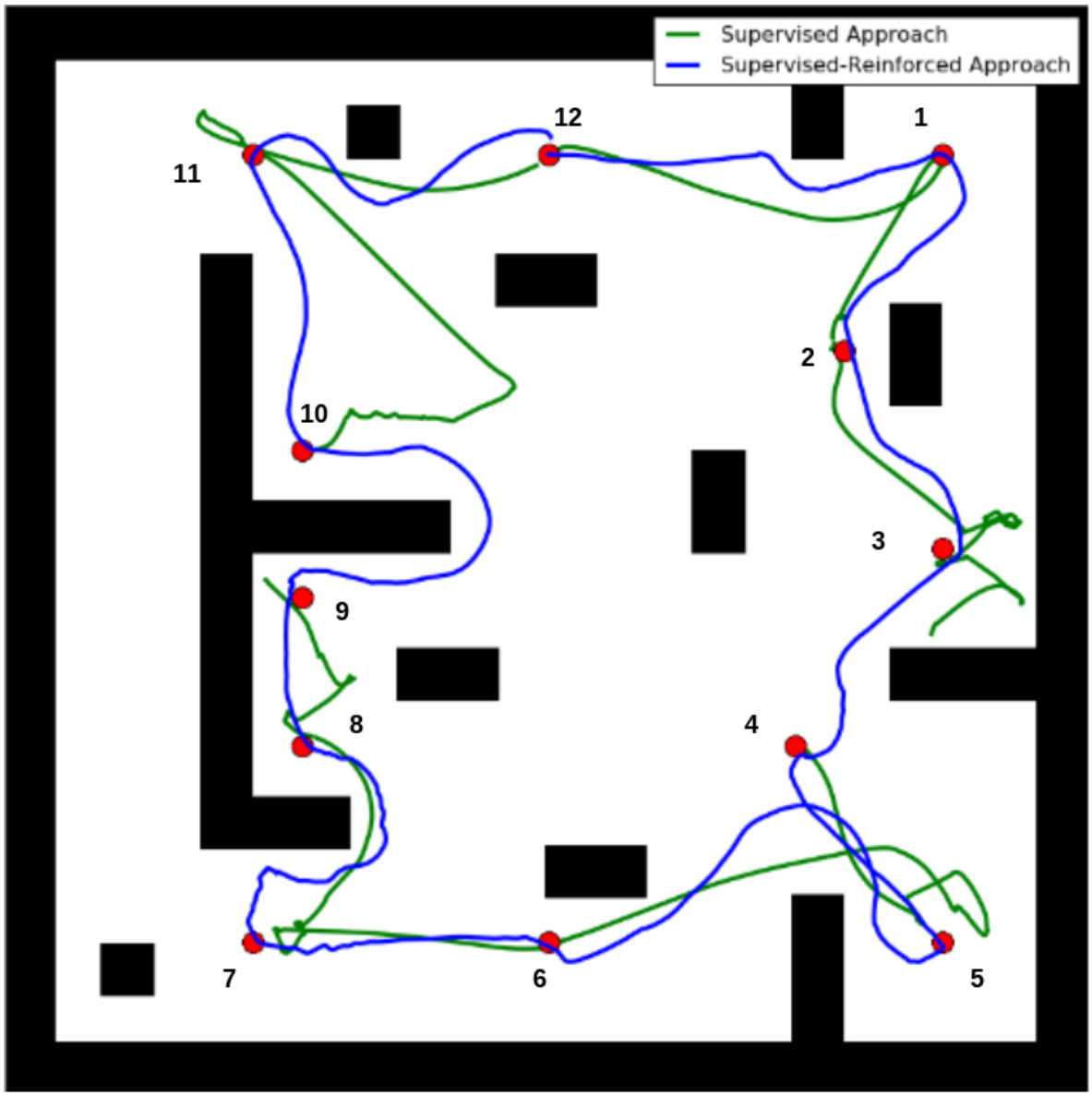


Figure 6-5: Trajectory comparison of Supervised Learning approach and Supervised-Reinforcement Learning Approach. While S-RL approach is able to complete the given trajectory, the SL approach fails to do so and struggles with time-outs and collisions around points $3, 7, 9$.

it's goal position and execute a recovery behavior. The deceleration behavior helps further improve the stability of the planner as the robot is comfortably able to reach a goal position within a stipulated time period. The results obtained on all the four test maps also show that this behavior helps make the planner collision proof. But this improved stability and collision avoidance ability does come at a cost as it is observed from the experiments that SR-L on an average has smaller linear velocity than SL.

The SR-L planner has also been able to overcome earlier issues, where if a goal was located close enough to the robot but was present at an opposite end of an obstacle lying between the two, the robot would usually struggle to reach it's destination as it had not been able to learn the behavior of navigating around the obstacle in such cases during it's training. But with S-RL, the planner is able to learn the required navigation behavior and is able to comfortably complete the trajectories that could earlier have ended in a collision or time-out. The planner's ability in this regard can be visualized from it's performance around the long corridor region on the left side of the map in Figure 6-4 and also on the trajectory comparison map in Figure 6-5. On the trajectory comparison map, during navigating from point 9 to 10, while the supervised learning planner struggles and eventually ends up in a collision, the S-RL planner is comfortably able to navigate around the obstacle and complete the required motion between the two points. Similar results are observed around points 3 and 5. The SL planner struggles in the transitions, 3 to 4 and 5 to 6 because of being surrounded by obstacles in a close proximity. But the S-RL planner faces no such issues during any of the transitions.

Even though the S-RL planner displays a good navigation behaviour around the long corridor in Figure 6-4 as mentioned above, all the time-outs listed in Table 6-1 also happen in this particular region. These time-outs usually occur if both the robot and the goal lie around the middle of the corridor region but on the opposite sides. In this particular case, the robot does try to navigate around the corridor in order to enter the goal containing region, but while attempting to do so, as the robot moves further away from it's initial position, it's Euclidean distance to the goal increases as well. The planner's aim is to reduce it's distance to the goal as the trajectory progresses. But in this case, since the corridor is quite long, the planner observes an increase in the distance for a significant amount of time as it attempts to reach one of it's ends. Not observing the required decrease in the distance, the planner abandons the motion towards the end of the corridor and instead begins to return to it's initial position where the Euclidean distance to the goal was the shortest. This results in an up-down motion of the robot through the corridor which eventually leads to a time-out. As stated earlier, this behavior is only observed if both the robot and the goal are at the opposite sides of the corridor and in the middle region. In all other cases, the robot is comfortably able to navigate to the opposite side of the corridor, majority of the time and reach it's goal position.

6-4 Key Takeaways

After observing the test results of the S-RL based planner, it can be concluded that the new planner is able to address the issues faced by the SL based planner. The S-RL planner is able to achieve a success rate of 100% on the test maps used for SL based planner and is also able to deliver similar results with 97% success rate on a much harder map. The new planner does not encounter even a single collision for all the 500 trajectories executed on the four test environments, making the new planner collision proof. It is also able to address the drifts and overshoots caused by the SL planner. Based on the results in this chapter, the final research question can be addressed as follows:

- **Does combining the Supervised Learning based approach with another learning methodology like Reinforcement Learning help in improving the performance of the proposed End-to-End planner?**
  It is observed that combining the Supervised Learning with Reinforcement Learning results in the rise of success rates of the planner along with a significant reduction in the crash and time-out rates. The stability of the planner also increases and it no longer suffers from issues like drifts and overshoots near the goal position. Hence it can be concluded that the combination of two learning approaches helps in improving the performance of the proposed End-to-End planner.

Chapter 7

Conclusions

This chapter summarises the whole thesis report and provides some suggestions for the future work.

7-1 Summary

In this thesis work, an End-to-End robot motion planner was designed using the simulation of Jackal UGV that could directly map raw sensor data like LiDAR scans to desired velocity commands. At first, the structure of the planner was proposed, which made used of LSTM cells in order to introduce temporal dependency between consecutive output commands of the planner. The planner was trained using Supervised Learning with the data collected in Gazebo simulation using ROS Move Base as the expert planner. Collecting the data was the biggest challenge during the thesis as the whole procedure was not only time consuming but also the whole performance of the planner was heavily dependent on the quality and quantity of the training data and the representation in which the training data was input to the network.

The planner was tested on various test maps and was able to achieve a success rate of 80% by having access to only robot's local information and no global information about the navigation environment. But the trained planner had it's flaws like sometimes overshooting and drifting around a goal position or not being able to avoid collision with an obstacle in time. In order to weed out these flaws, the Supervised Learning approach is combined with Reinforcement Learning. Combining two different methods came with it's own set of challenges like finding a new simulator to meet the demands of scaling up the real time collection of the training data.

The new updated planner is observed to deliver far superior performance compared to it's previous version, achieving a success rate of nearly 100% on different test maps and also delivering a crash rate of 0%. The new planner was also able to improve it's exploratory behavior and was able to navigate around the obstacles in order to successfully complete it's trajectory.

7-2 Future Work

Resulting from the thesis work, some recommendations are made for the future work as follows:

- **Implementation on a real robot:** The proposed planner has been trained and tested in a simulation. It will be interesting to see how the planner goes about executing a desired trajectory in a real world environment where the input sensor readings contain much more noise.
- **Implementation in a Dynamic Environment:** Currently the planner has been trained to execute an autonomous navigation behavior in a static environment. Training it to safely navigate in the presence of moving obstacles like pedestrians in a corridor presents a great scope for research
- **Increasing the scope of End-to-End approach:** It will be intriguing to see if instead of velocity values, a network can learn to directly map raw sensor readings to the actuator commands, thus making the planner more end-to-end.
- **Experimentation with other experts:** Another potential research direction would be to train the network with a different expert like a Model Predictive Control based planner and instead of training the network to output a single command, train the network to output a series of commands over a prediction horizon and visualize the robot's predicted trajectory at each time step.
- **Further improvement in Supervised-Reinforcement Learning End-to-End Planner:** It would be ideal to work further on improving the exploratory nature of the planner so that it can comfortably navigate around a fairly large obstacle to reach the goal at an opposite end.
- **Sensor Fusion:** Currently the planner only relies on the LiDAR data to make it's predictions. It would be interesting to see if fusing the laser data with that of other sensors like RGB-D cameras can further enhance the performance of the End-to-End planner.
- **Expanding the research to other robots:** Currently the planner solely focuses on the differential drive ground based mobile robots. An interesting research direction would be to train the planner to be robust enough to be able to function on other types of robotic platforms as well like quad-rotors or autonomous boats.

Appendix A

Other Network Architectures

This chapter discusses the other networks considered for the research and the reasons that influenced their candidacy.

A-1 Autoencoder

An Autoencoder is a Machine Learning model that consists of two parts, an Encoder and a Decoder. The Encoder takes as input $x \in R^N$ and maps it to a latent space, $h \in R^{N'}$. This mapping is achieved using a deterministic function of the form, $h = \sigma(Wx + b)$. Here, σ is an activation function and W, b are learnt parameters to obtain the required mapping. The Decoder then uses the obtained function to reconstruct the original input using a reverse mapping, $h' = \sigma(W'h + b')$. The learnt parameter sets are usually constrained to be of the form, $W' = W^T$, resulting in same weights being used to encode the input and reconstruct it using it's latent representation [19].
Using the Occupancy Grid maps as input data of size 60x60, an Autoencoder consisting of convolution layers is trained to reconstruct the input maps. Achieving quality reconstructions of the Occupancy Grid maps results in the Encoder part of the network extracting relevant information and learning a representation of these maps that is robust and deterministic in nature. The designed Autoencoder model is based on the network presented in [2] (see Figure A-1).
The model is trained over 60,000 iterations with a learning rate of 0.005 and batch size of 16. The Encoder part of the network consists of three convolution layers and one fully connected layer. The fully connected layer is obtained by flattening the output of the third convolution layer and consists of 512 units. This fully connected layer acts as the input to the latent space of dimension 64. The Decoder is the mirror image of Encoder and consists of three transpose convolution layers. The details regarding certain hyper-parameters associated with each convolution and transpose convolution in the whole model are presented in Table A-1. The cost function deployed to train the Autoencoder is Mean Squared Error (MSE).

$$MSE = \frac{\sum_{i=0}^{N} \sum_{j=0}^{N} (x_{ij} - y_{ij})^2}{2N}$$

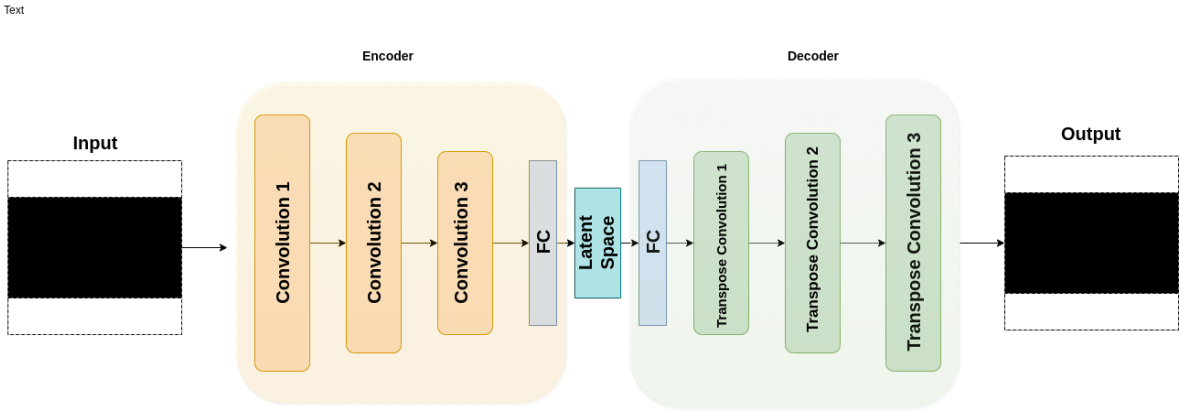


Figure A-1: Autoencoder Architecture

Table A-1: Specifications of convolution and de-convolution layers of the autoencoder

Layer	Kernel Size	Filters	Strides	Activation
Convolution 1	5 x 5	64	2	Relu
Convolution 2	3 x 3	32	2	Relu
Convolution 3	3 x 3	8	2	Relu
Transpose Convolution 1	3 x 3	8	2	Relu
Transpose Convolution 2	3 x 3	32	2	Relu
Transpose Convolution 3	5 x 5	64	2	Relu

Where, x_{ij} and y_{ij} represent each value in the input and reconstructed array respectively. In Figure A-2, some of the results of the trained Autoencoder model are presented. Since, the trained model provides with good reconstructions of the input data, the encoder part of the model can be used in designing the required End-to-End architectures and extract relevant information from the input data.

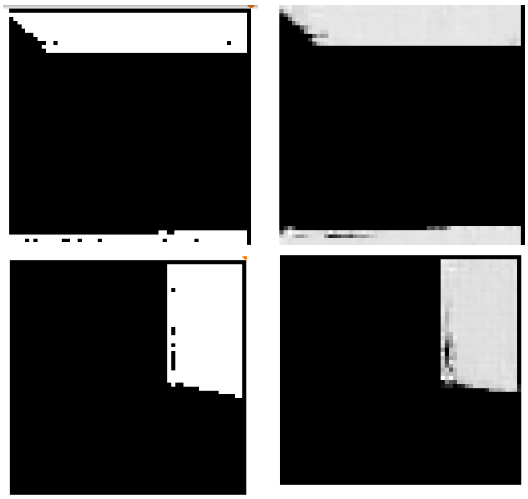


Figure A-2: Left Column: Original occupancy grids; Right Column: Their corresponding reconstructions using the Autoencoder

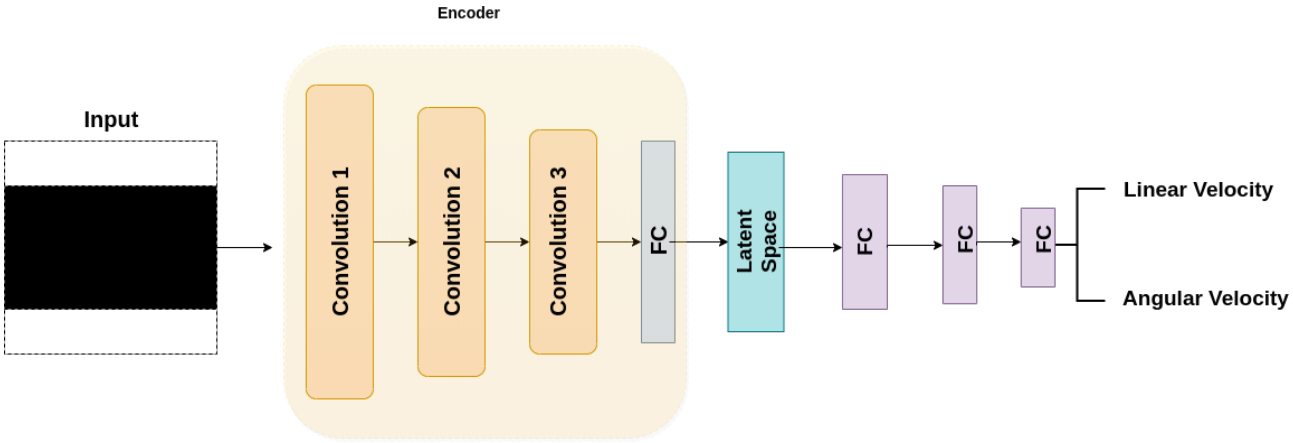


Figure A-3: Encoder connected to fully connected layers

A-2 Encoder with Fully Connected Layers

Using the Encoder part of the Autoencoder, a model is trained with the Occupancy Grid data as the input. The output of the latent space is fed to two fully connected hidden layers of size 32 and 16. The final layer of the model of size 2 outputs the linear and angular velocity for the robot (see Figure A-3).

The model is trained over 20,000 iterations with a learning rate of 0.01 and batch size of 16. During the training process, only the fully connected layers connected to the Encoder are trained and the Encoder makes use of the pre-trained weights derived from the training of the Autoencoder in the previous section. The loss function used for training is MSE:

$$MSE = \frac{(Y - \hat{Y})^2}{2}$$

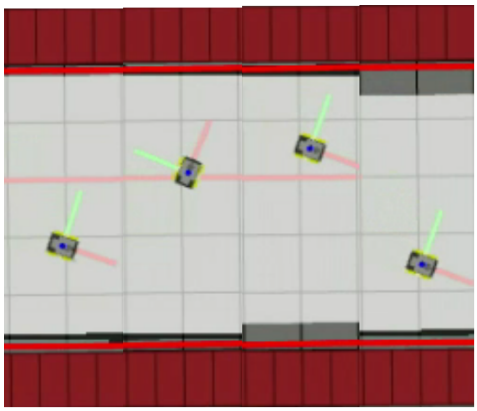


Figure A-4: Network performance with fully connected layers

Here, Y represents the reference output from the training data and $\hat{Y}$ represents the velocity predictions from the network. Figure A-4 provides some snapshots from the performance of the trained model. It is observed that the resulting robot motion is highly unstable with the robot executing a zigzag motion between the two ends of the simulated corridor and it eventually ends up colliding with one of the simulated walls.

A-3 Adding LSTM Blocks

As observed in the implementation of the previous model, the resulting robot motion is quite unstable. Ideally, the desired robot motion is required to be much more smoother. This smoothness/stability is achieved with the introduction of temporal dependency in the network by the use of LSTM blocks as discussed in chapter 2. The size of the input occupancy grid and the latent space is also increased to 120×120 and 128 respectively in order to increase the amount of visual information being input to the network (see Figure A-5).

The model also has a LSTM channel for the goal information. Two different candidate configurations are considered in which the goal information is sent to the model. They are:

- The robot's relative position to the goal in Cartesian coordinates
- The robot's relative position to the goal in polar coordinates

During tests, it is observed that the network with the polar goal information outperforms the one with the goal information in the form of Cartesian coordinates. The major reason being that the polar coordinates also provide the angular differences between the goal and the robot, which help the network decide the navigation direction.

But even with polar goal information, the network only achieves a success rate of around 27%. The possible reason for this result lies in the fact, the way in which the visual data is input to the network. The occupancy grids, despite providing a 2D representation of the robot's surroundings, are binary in nature. If there is an empty space, it is labeled as 0 and an obstacle is labeled as 1. They don't contain the true values of distance between an obstacle

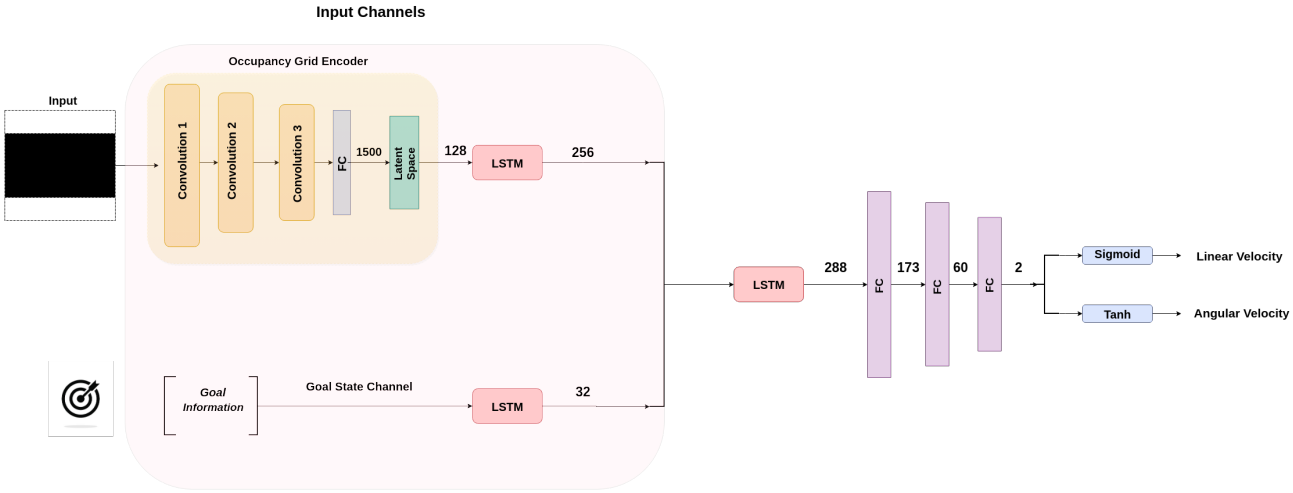


Figure A-5: Model with input channels consisting of occupancy grid and goal information

and the robot. Also, these grids are centered around the robot at an angle of 0 degrees and contain no information regarding the orientations of the robot and the obstacles. All these factors make it hard for the network to decide upon the suitable control commands in order to navigate in an obstacle filled environment. This in turn leads to the selection of raw 2D laser scan readings as the input data and other network choices as detailed in chapters 2 and 3.

Bibliography

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," pp. 1–9, 2016.

[2] M. Pfeiffer, G. Paolo, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "A Data-driven Model for Interaction-aware Pedestrian Motion Prediction in Object Cluttered Environments," 2017.

[3] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car," pp. 1–8, 2017.

[4] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "LIDAR-based Driving Path Generation Using Fully Convolutional Neural Networks," 2017.

[5] A. H. Qureshi, M. J. Bency, and M. C. Yip, "Motion Planning Networks," 2018.

[6] T. Lei and L. Ming, "A robot exploration strategy based on Q-learning network," *2016 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2016*, pp. 57–62, 2016.

[7] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation," pp. 1–6, 2017.

[8] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, L. Fei-fei, and A. Farhadi, "Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, no. 1, pp. 3357–3364, 2017.

[9] J. A. Bagnell, "An invitation to imitation," Jun 2018.

[10] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile Off-Road Autonomous Driving Using End-to-End Deep Imitation Learning," 2017.

[11] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," *CoRR*, vol. abs/1011.0686, 2010.

[12] J. Zhang and K. Cho, "Query-Efficient Imitation Learning for End-to-End Autonomous Driving," 2016.

[13] "Cs-230 convolution neural network cheat sheet, available at https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks."

[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, 2015.

[15] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced Imitation: Sample Efficient Deep Reinforcement Learning for Map-less Navigation by Leveraging Prior Demonstrations," 2018.

[16] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1889–1897, 2015.

[17] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," *34th International Conference on Machine Learning, ICML 2017*, vol. 1, pp. 30–47, 2017.

[18] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, no. 2-4, pp. 189–208, 2008.

[19] J. Masci, U. Meier, and D. Cire, "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," *International Conference on Artificial Neural Networks*, no. June, pp. 52–59, 2014.

Glossary

List of Acronyms

GPU	Graphics Processing Unit
CNN	Convolution Neural Networks
LiDAR	Light Detection and Ranging
ROS	Robot Operating System
ReLu	Rectified Linear Unit
i.i.d	Independent and Identically Distributed
LSTM	Long Short Term Memory
SL	Supervised Learning
S-RL	Supervised Reinforcement Learning
MSE	Mean Squared Error
MAE	Mean Absolute Error
RL	Reinforcement Learning
CPO	Constrained Policy Optimization
TRPO	Trust Region Policy Optimization