People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Power and Control

Project Report Presented in Partial Fulfilment of

The Requirements of the Degree of

'MASTER'

In Electrical and Electronic Engineering

Option: Control Engineering

Title:

End-to-End Learning-based Navigation of Autonomous Mobile Robot

Presented By:

MEHRAB ANIS ABDELDJALIL

Supervisor:

Dr. GUERNANE

Registration Number:...../2020

ABSTRACT

In this work we present an end-to-end learning approach that is able to perform target-oriented navigation and collision avoidance using Deep Neural Network. This approach can be defined as learning a model that maps sensory inputs, such as raw 2D-laser range findings and a target position, to navigation actions for controlling the mobile robot such as steering commands. Compared to the traditional autonomous navigation systems, which often require perception, localization, mapping, and path planning, the end-to-end learning approach offers a more efficient method. which utilize large set of expert navigation demonstrations to learn the desired navigation policy.

The end-to-end learning approach has gained considerable interests in autonomous navigation in academic and industrial fields. Researches have already used different artificial neural networks to predict steering commands. However, most of the existing end-to-end methods are used for lane keeping for self-driving cars. therefore, we propose an end-to-end navigation model for mobile robots that is based on a Convolutional Neural Network (CNN). The network was trained using expert demonstration data which was generated in virtual simulation environments. The learned model was test in real time simulation and gave an acceptable result, however, it suffered when it encounters situations that requires hard maneuvers. Therefore, in order to overcome some of these difficulties, we proposed an improved model which incorporates the temporal information in the prediction process using the Long Short-Term Memory (LSTM) network. basically, this model aims to include the motion history of the robot in the steering prediction model. The improved model showed its ability to predict steering commands with high performance compared to the expert operator. However, this model imposed some limitations which will be further discussed in this remaining parts of this thesis.

ACKNOWLEDGEMENTS

I would firstly like to thank my advisor, Dr. Reda Guernane, for his support and guidance throughout this thesis. Also, I must express my very profound gratitude to my parents, my siblings, and all my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Autonomous Navigation

The evolution of robotics industry has shown an intensive growth in the previous years. Mainly due to the necessity to replace human intervention in various situations, Such as: military missions, dangerous explorations, delivery services, health care, etc. For this reason, researchers aimed to solve this problem by trying to make robots do those human tasks autonomously.

Robots come in many different shapes and can perform diverse tasks. The most common distinction, is between *fixed* and *mobile* robots. Fixed robots are mostly industrial robotic manipulators that work in well-defined environments adapted for robots. Whereas, mobile robots are expected to navigate around and perform tasks in large, uncertain environments that are not designed specifically for robots. They need to deal with situations that are not precisely known in advance and that change over time. Such environments can include unpredictable entities like humans and animals. To date, most of the navigation systems used to deal with these situations can be categorized into two main classes: traditional autonomous navigation systems, and end-to-end autonomous navigation systems.

1.1.1 Traditional autonomous navigation system

The traditional autonomous navigation system can be divided into four main components [1], as shown in Figure 1.1. Similar to human vision, the Perception component uses sensors to continuously scan and analyze its surrounding environment. This component usually consists of functions for obstacle detection and tracking. The Localization and Mapping module is where the robot creates a map of the environment, and localize itself relative to that environment. The path planning uses the information provided by the previous two modules to create a safe collision free path for the robot Controller to execute. Finally, the Robot Control module determines the steering commands, such as rotational and angular velocity, to drive the robot along the planned path.

Although a significant progress has been made in developing the traditional navigation systems, there still many challenges towards building a fully autonomous navigation system. However, because the traditional approach decomposes the autonomous navigation problem to a

hierarchy of sub-problems, where each one has its own optimization problem, Errors can accumulate from previous processing stage to next stage, leaving the final result inaccurate. These drawbacks have inspired research into the end-to-end learning approach for autonomous navigation that does not require manual decomposition of the autonomous navigation system.

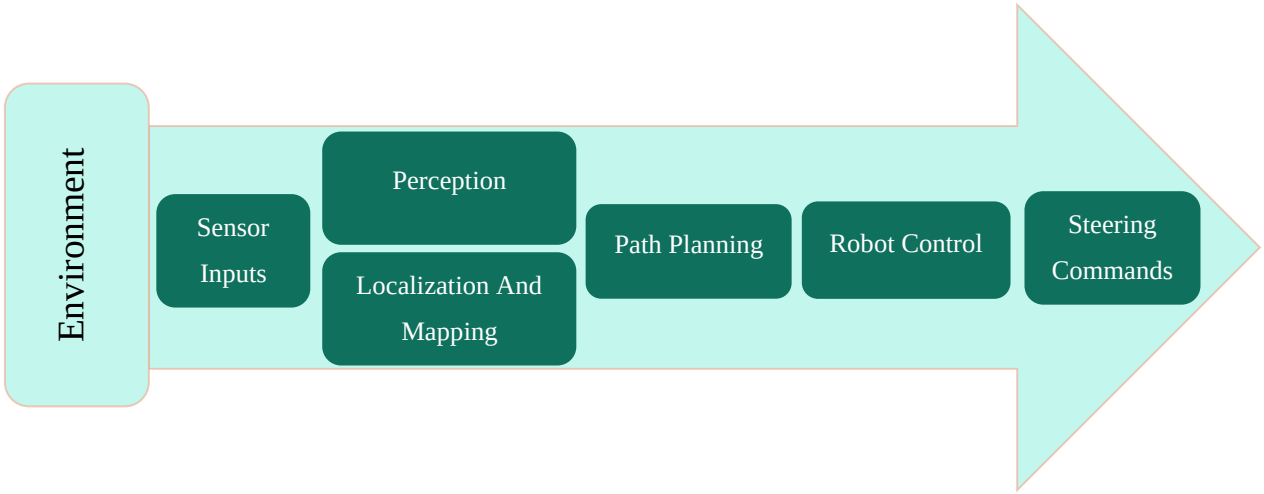

Figure 1.1: Overview of a traditional autonomous navigation system

1.1.2 End-to-End autonomous navigation system

In contrast to the traditional approach, the end-to-end navigation systems can directly output the linear and angular velocities of the robot from sensor inputs in a single step. Furthermore, these types of systems are self-contained that can carry out all the processes automatically from mapping based on sensory inputs, such as a LIDAR scanner, to the actions necessary for navigation. Usually, the end-to-end systems are designed to learn from expert demonstrations rather than manually-designed rule based modules.

In this report, we propose a single-step, end-to-end, learning-based navigation approach for mobile robots, which directly infer the final steering commands from raw data inputs. The system is trained on Data that was collected while an expert operator was driving the robot, with each data frame contains 2D lidar scans, target location and steering commands. The trained model gives steering commands for navigation as output in real-time making it possible to be implanted on mobile robots.

Figure 1.2: Overview of end-to-end autonomous navigation system

1.2 Thesis Structure

The organization of this thesis is as follows; Chapter 2 introduces the first approach to solve the end-to-end navigation problem for mobile robot. chapter 3 demonstrates the improved end-to-end model. Chapter 4 gives the experiment results and discussion. The final Chapter address the conclusion about this work and the future work.

Chapter 2: End-to-End Navigation Model

One of the major challenges in robotics is to make robots perform as desired by human operators. Regarding ground robot navigation, this problem is defined as getting the robot from the current position to a target position fulfilling the desired navigation policy. Although objectives like, e.g. short path or a safe distance to obstacles are perfectly clear to the human operator, it typically requires time-consuming hand tuning, such that, the robot moves as desired and as required. Additionally, classical navigation solutions require several steps of data preprocessing that typically are decoupled. A map of the environment has to be provided, the sensor data has to be preprocessed and potential objects have to be detected such that the navigation algorithm can react accordingly in a later stage.

2.1 Approach

The key idea of the end-to-end learning approach is to combine all the previous sub-problems to a single one optimization problem. simply by treating it as a data-driven machine learning problem, in which the used data corresponds to the desired navigation behavior of the mobile robot. To do this, a neural network model was trained on data that was collected when a expert operator was driving the robot in virtual simulation environment. Basically, the network takes the lidar scan and relative target location as an input and tries to predict a suitable steering command, which takes the robot towered the goal while avoiding obstacles.

Data was collected in several virtual simulation environments, in which the robot was being driven by a human operator. Furthermore, while robot was moving, data was being collected at frame rate of 5Hz, with each frame contains a 2D lidar scan, target location, and steering commands. This data was used later to train the neural network by giving it the lidar and target data as an input, and optimize its parameters so that the difference between the predicted and the actual steering commands is minimal. The main advantage of using neural network for this specific task, is its ability to generalize the driving behavior to environments which has not seen before.

Mainly, we are trying to find the mapping function F that takes as an input $x = [x_{lidar}, x_{target}]$ and outputs the steering command $u = [v, \omega]$. as shown in Figure 2.1.

$$u = F_\theta(x) \tag{2.1}$$

This function can be obtained by applying the back-propagation optimization technique to optimize the neural network parameters(weights) through a set of iterations on the collected dataset. Throughout this process, the network adjusts its weights θ so that the difference between predicted and actual steering commands is minimal.

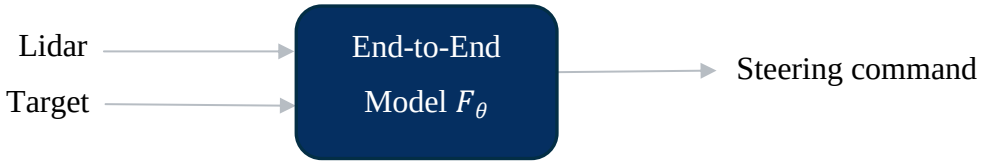

Figure 2.1 *End-To-End Model*

The mapping function can be of any appropriate form; in this approach we used a Convolutional Neural Network (CNN) as depicted in Figure 2.2. this network uses the lidar 2D 360 point clouds to extract the spatial features in the environment, alongside with the target location to predict steering commands that ensures no collision till the robot reaches the target location.

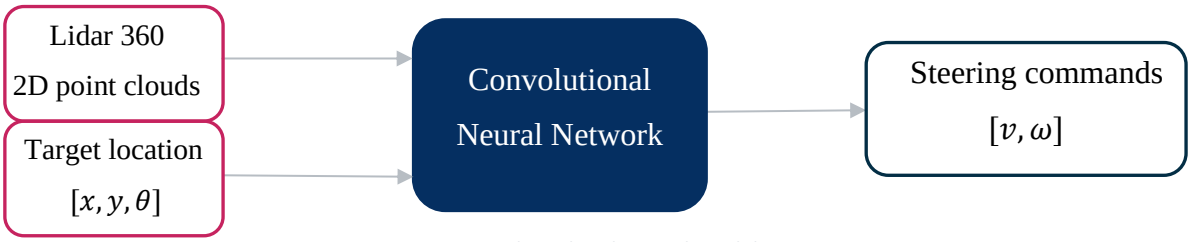

Figure 2.2 CNN-based end-to-end Model

2.2 Related Work

Traditionally, robot navigation had been performed in a multi-step process. The robot would first detect obstacles, draw a map, localize the robot, and plan the robot's local path according to environmental structure. Nonetheless, the end-to-end methods have been introduced showing comparable accuracy in mobile robotics navigation with the sensor-based multi-step methods.

In 1989, a three-layered neural network was trained for the task of lane following of an autonomous vehicle called ALVINN (Autonomous Land Vehicle In a Neural Network) [2]. this vehicle had been driven under controlled field without any human intervention, and the network used 30x32 image and a laser range finder as an input to generate a steering direction in order to keep the vehicle in the lane. Although, it was one of first examples of an autonomous vehicle using the end-to-end learning approach. nevertheless, its success suggested the potential of neural networks for autonomous navigation.

Later on, in 2005, an obstacle avoidance method based on end-to-end approach was applied on a mobile robot called DAVE which is presented in [3].A Convolutional neural network was trained on data that was collected in off-road situations. the trained network could directly predict steering angles from input images of the robot's point of view to avoid obstacles. Later on, in 2016, an approach was pioneered by researchers at Nvidia with their paper "End to End Learning for Self-Driving Cars" [4]. Where they took as an input the raw image data and attempt to output throttle, break, and steering commands. By learning, once again, from human driving commands in an imitation learning approach.

Regarding mobile robotic applications of end-to-end learning, the work of [5], presented an approach that learns a left/right controller for an unmanned aerial vehicle (UAV) based on image data. The UAV was able to autonomously navigate through a forest while successfully avoiding collisions with trees in the majority of the cases. However, only the left/right motion has to be controlled while the forward motion command is still selected by a human operator. With this approach the robot can drive reasonable paths, however no specific goal can be reached. Also, in the work presented in [6], they used a lidar data instead of camera, also they used a CNN based model to predict the steering command seeking a specific target.

2.3 Preliminaries

2.3.1 Virtual Mobile Robot: TurtleBot

TurtleBot is a two wheeled, programmable, ROS-based mobile robot with open-source software [7], which is used for education and research purposes. For this experiment we will use the Turtletbot3 software package, that provides virtual robot which can operate in a virtual simulator like gazebo. More specifically, the Turtlebot3 burger type shown in Figure 2.3 will be

used as our virtual mobile robot, which is used for both data collection and testing the trained model.

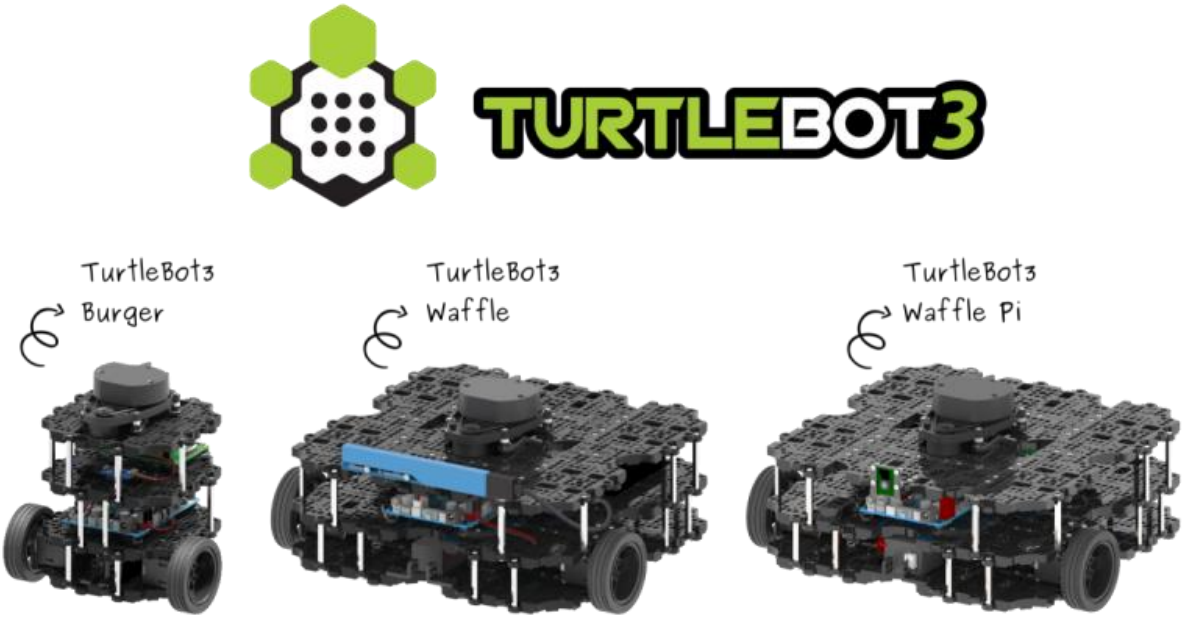

Figure 2.3 TURTLEBOT3 mobile robots

Turtlebot3 burger: Sensor suit

The main sensor that is of interest to our navigation algorithm is the 360° LIght Detection And Ranging sensor (LIDAR). As shown in Figure 2.4. basically, The LIDAR sensor mounted in the turtlebto3 is a 2D laser scanner capable of sensing 360 degrees that collects a set of 2D data point clouds around the robot , as shown in Figure 2.7, to use it for Navigation purposes.

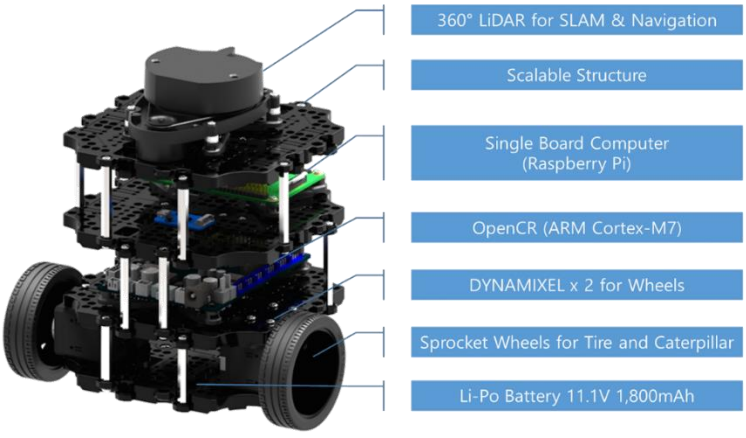

Figure 2.4 TURTLEBOT3 burger sensor suit

Turtlebot3 burger: motion model

Generally, Robot motion can be modeled either by considering the geometric constraints that defines its motion, or by considering all of the forces and moments acting on the robot. The first case is known as kinematic modeling. While the second is known as Dynamic modeling. At low speed when the accelerations are not significant, kinematic modeling is more than sufficient to capture the motion of the robot. For high speed, Dynamic modeling is more suitable, and it can do a great job in estimating the robot motion throughout the whole robot operating range.

The Turtlebot3 is a two wheeled differential drive mobile robot. Its motion is constrained to move forward because its wheels points only in the forward direction. And this type of constraint is called a nonholonomic constraint. Which means it restrict the rate of change of the position of our robot. So, our robot can roll forward and turn while rolling, but it cannot move sideways directly. this constraint can be used to define a kinematic model of the Turtlebot3 robot.

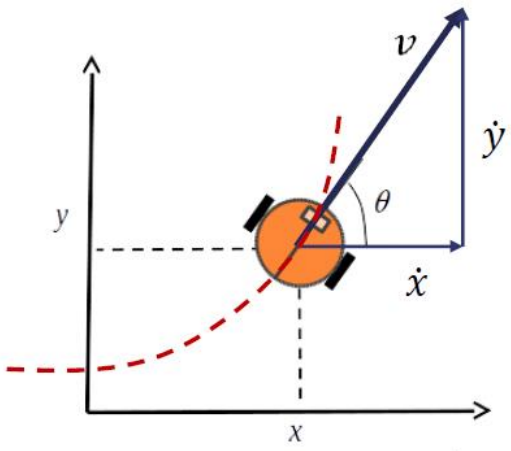

Figure 2.5 two wheeled robot motion

The velocity of the robot $\boldsymbol{v}$ is defined by the tangent vector to its path. The orientation angle is defined as θ. And from Figure 2.5 we get:

$$\frac{dy}{dx} = \tan\theta = \frac{\sin\theta}{\cos\theta} \tag{2.2}$$

By rearranging the above equation, we get the Nonholonomic constrain equation:

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0 \tag{2.3}$$

The motion of the robot is then defined by these equations:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{2.4}$$

Therefore, this model takes as an input the forward and rotational velocity $[\boldsymbol{v}, \boldsymbol{\omega}]$, and represents the robot motion model using a vector of three states presented in equation (2.4), that generate the next state which represent the XY position of the robot and is heading θ. As shown in Figure 2.6.

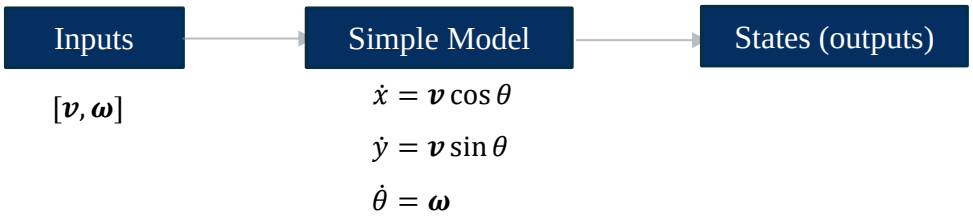

Figure 2.6 *two-wheel robot simple model*

Remark: the developed navigation algorithm relies on this kinematic model by giving it the predicted velocities $[\boldsymbol{v}, \boldsymbol{\omega}]$ to control the robot.

2.3.2 Simulation Environment

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. one of those simulators is Gazebo [8], which is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs.

Gazebo gives the ability to construct 3D environments that simulate realistic and difficult scenarios. Across these environments, the navigation algorithm was trained and test. To provide a realistic obstacles situation, obstacles where added and removed occasionally, meanwhile the robot was driven across these environments. An shown in Figure 2.8. these obstacles may be a chair, table, standing person, etc.

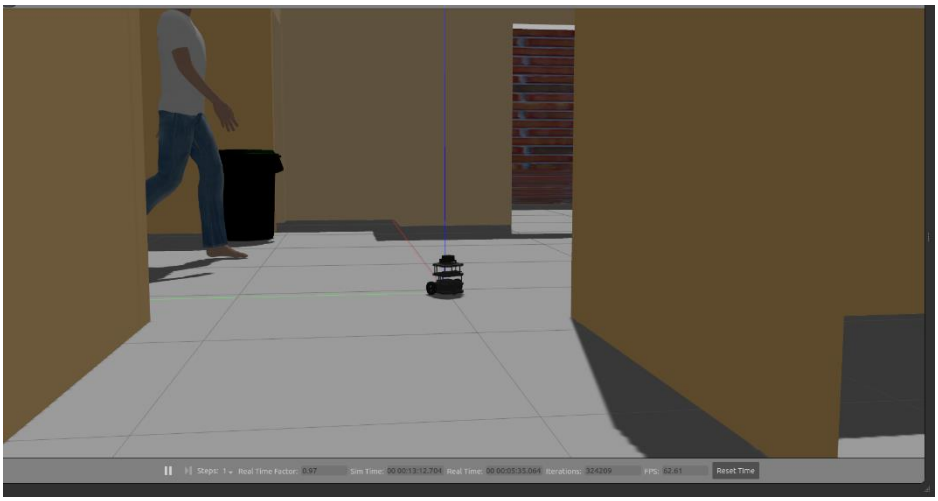

Figure 2.8 Gazebo simulator| Turtlebot navigating in an indoor environment

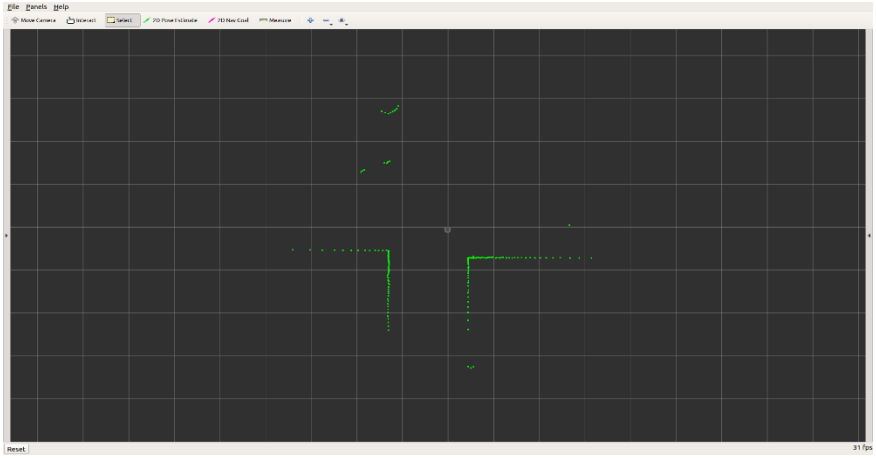

Figure 2.7 Visualization of the 2D point clouds (in yellow) generated by the Turtlebot's LIDAR

2.4 Data Collection

A crucially important requirement of data collection process is to collect a large amount of data with enough diversity of terrain and obstacles. Basically, the robot was driven for more than 3 days of time across eight different indoor environments, each of which has different characteristics. Throughout this process, a sperate module was collecting data of 2D lidar scanner, target location and velocity commands.

The lidar scan data was collected at a rate of 5Hz, while the velocity command and target location were collected at rate of 20hz. It is important to know, the lidar and target data was recorded relative to the robot reference frame. as the network will be trained on data that is

collected from robot's perspective. After data has been recorded it was synchronized based on the data header time stamp, which resulted in a dataset of 250,000 frames.

At first, we controlled the robot in the simulator using a Joystick Gamepad controller. However, we couldn't control the robot smoothly through the environment, as we kept hitting obstacles along the way, and this may lead to a bad dataset. Therefore; the Dynamic Window Approach (DWA) alongside with the A* algorithms were used as an expert operator which drives the robot instead. Although; It was better if the robot was driven by a human operator, and it would have been even better for the data to be collected from real world robot, where sensors readings translate the actual characteristics of the environment.

Data preprocessing

The lidar scans the whole horizonal surface collecting data at rate of 5hz, with point density of 360 points per scan. Also, because laser range data has "inf" values, which correspond to no *information region*, and by taking into account that 'inf' values cannot be propagated through the network. it was replaced with "-1". The reason we choose this value, is because it works with our network the best among those that we previously tried, such as {"0.0","10.0","100.0"}.

The target information contains the XY position as well as the heading θ: $x_{target} = [x, y, \theta]$
In order for the target position to be in the robot range of view, the target position was chosen to be in the circle surrounding the robot. As shown in Figure 2.9

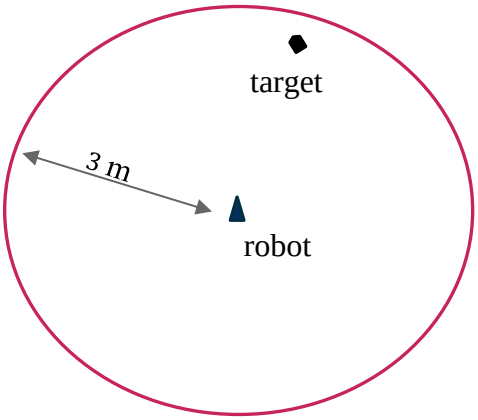

Figure 2.9 target to robot range

Due to the low scanning rate of lidar (5hz) in the simulation environment, the maximum translation velocity of the robot was adjusted to a lower value. So that no important features are missed during driving.

2.5 Model Structure

The mapping between the input data and the output steering commands may result in a very complicated model among machine learning approaches. Convolutional Neural networks (CNN) are well known for their ability to model a complex and non-linear dependencies.

First, we will use the Convolutional Neural Network as a feature extractor for the LIDAR data. Second, Unlike the model proposed in [6], the target position and heading will be fed to a series of dense layers that is responsible for up-sampling the target information to higher dimension features. And finally, the output of those two networks will be fed to a series of fully connected layers that outputs the steering command. As depicted in Figure 2.10.

The CNN is able to successfully capture and extract features from the environment by processing the lidar data that will help in understanding the scene. The network configuration was designed through a series of experiments taking into account both the training and validation losses. The network architecture consists of nine convolutional layers, that were designed to perform feature extraction. We used "same" convolution across all layers. And we let the dimension reduction to the Max Pooling layers. The first three convolution layers was used with kernel size of 7×1, the next two convolutional layers used with kernel size of 5×1, while a kernel size of 3×1 is used for the remaining convolution layers.

The target information is given as an input to a series of dense layers, which outputs a feature with dimension of $(1, 512)$. This feature is then concatenated with the flattened lidar features provided by the CNN that has an output of dimension of $(22 * 256)$. Later on, the concatenated features are propagated through a series of fully connected layer.

It is important to note, the fully connected layers are designed to function as a controller for steering commands, although, by training the end-to-end system, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serves as controller.

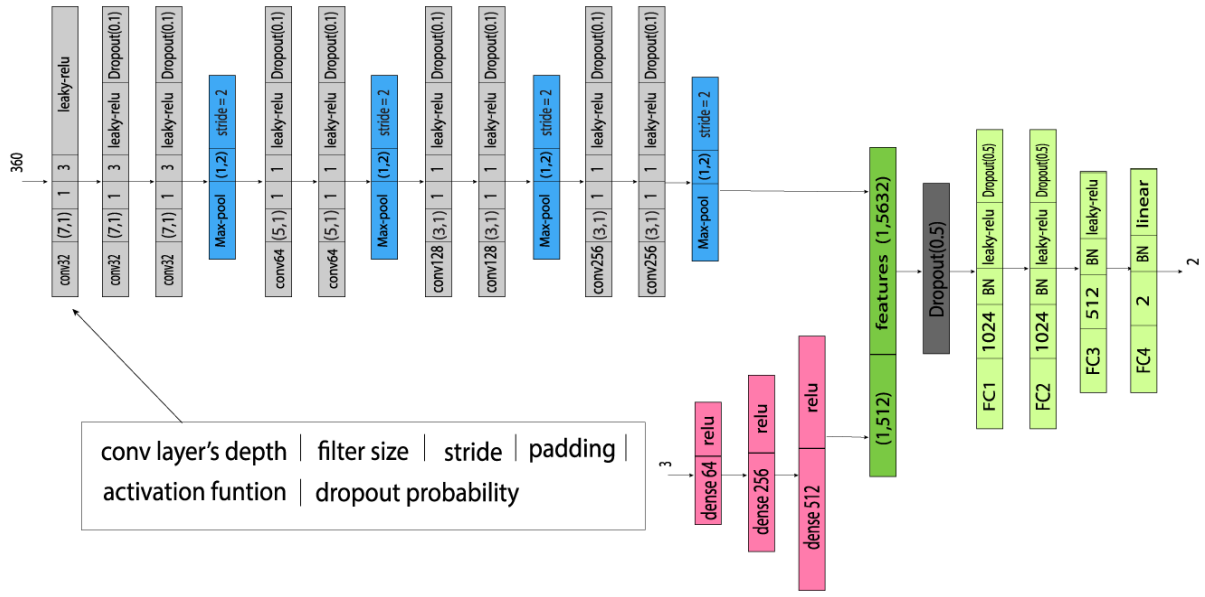

Figure 2.10 Convolutional Neural Network (CNN) Architecture. inputs: lidar data (360) and target location (3), output: velocity commands (2)

2.6 Training details

The collected data was divided to a train and validation datasets. The validation dataset corresponds to a 25,000 data sample that has been collected from an environment which is used only to test the trained network. in the other hand, the training data corresponds to 225,000 data samples, that were collected from eight other environments. All of the data was normalized on the fly using the formula (2.5).

$$x_{norm} = \frac{x - mean(x)}{std(x)} \tag{2.5}$$

Learning to predicted the steering commands can be considered as a regression problem. For this reason, we adopted a simple form of mean squared error as an objective function which will be minimized through back-propagation of the network.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_{predicted}^{[i]} - y_{actual}^{[i]}\right)^2 \tag{2.6}$$

The Training scheme is shown in Figure 2.11, where the CNN was trained for six hours, with a stochastic gradient-based optimization method called Adam [9], which automatically sets the relative step sizes of the parameters based on the local curvature of the MSE loss(error). The optimizer learning rate started at 0.001 and then reduced based on loss values at each iteration.

Also, since the GPU memory cannot hold training data all at once, the training data was divided into batches, each of which is equal to 128. Where the network loads the data batch by batch until it completes one loop over the whole dataset. The training and validation losses are shown in Figure 2.12

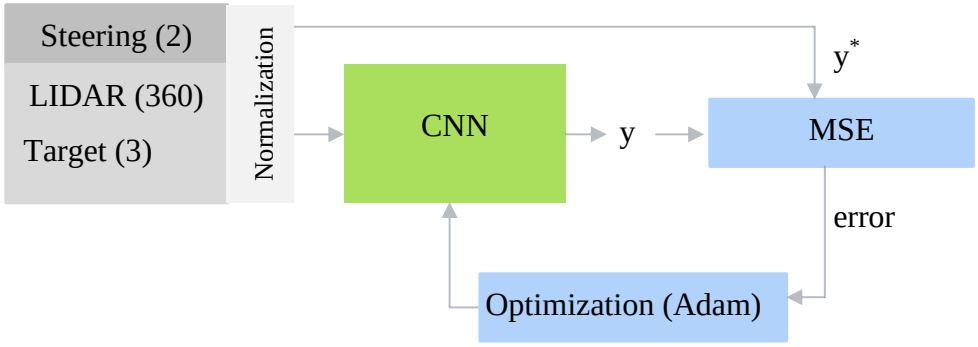

Figure 2.11 Training the neural network

During training, the network tends to overfit starting from 10th iteration. Therefore, Dropout layers have been added mostly across the whole network. Also, L2 regularization term has been added to the loss function. These two regularization methods slower the learning curve, but helped the model to generalize a little bit better. So, the validation loss was much better in terms of convergence. Although the network was iterated through the train dataset for 60 times, we used the model saved at epoch(iteration) number 16, just before the model started to overfit. Nevertheless, if the network was trained for enough time, the train loss will converge to zero, in contrast to the validation loss which will not converge.

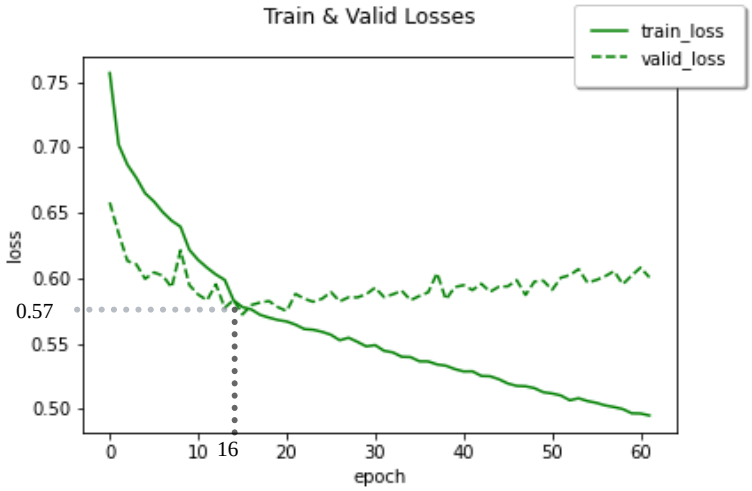

Figure 2.12 Train and Evaluation losses

During the training stage, one important issue needs to be addressed, which corresponds to the data used for training that is highly unbalanced as shown in Figure 2.13. where the most of the steering commands data corresponds to low rotational commands. Therefore, the trained model based on these unbalanced data may tend to navigate straight while still have low losses. As depicted in Figure 2.14; the network tends to learn the translation velocity much better than the rotational one. This is due that fact that, the dataset was collected while the robot navigating to the target in a straight line more often than the robot avoiding obstacles. Therefore, this issue will make the avoiding obstacles task much harder.

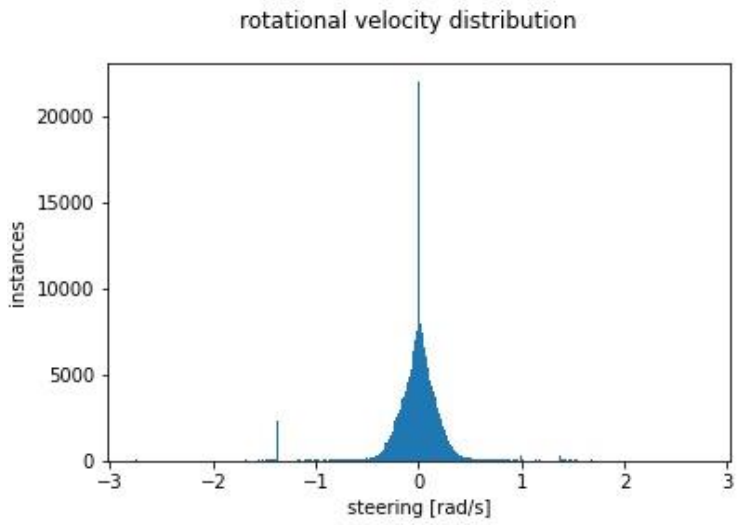

Figure 2.13 Histogram of rotational velocity commands in training data.

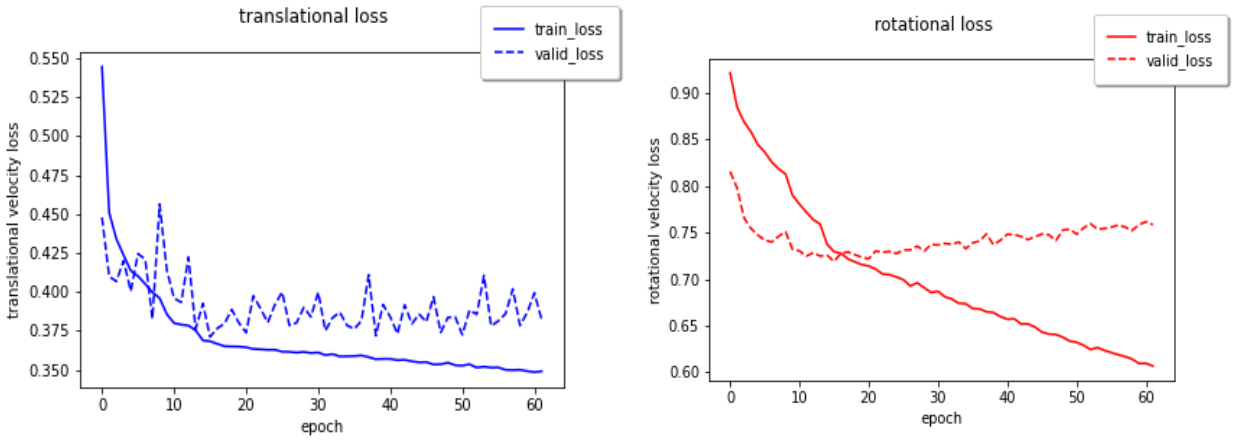

Figure 2.14 Translational and Rotational command Velocity losses

In general, through the process of collecting data, training, and testing. The more data we collect the better the model is able to generalize (the lower the validation loss will be). Although,

during this period, we collected about a quarter of a million of data samples, it was not enough. And based on the rate of improvement of the model when increasing the dataset size. We can say that, the model needs more than one million of data samples to be able to generalize better.

2.7 Evaluation

The model was evaluated using three different small datasets, each of which was collected from different environment. an input/output tuple has been generated from each dataset. One from sub dataset used for training(train), and one from the validation dataset which used for validating the model(eval1). Whereas the evel2 dataset was not used for either training or validation. The error between the translational and rotational steering commands between the ones generated by the network and the ones generated by the DWA algorithm was computed for each tuple.as depicted in Figure 2.15.

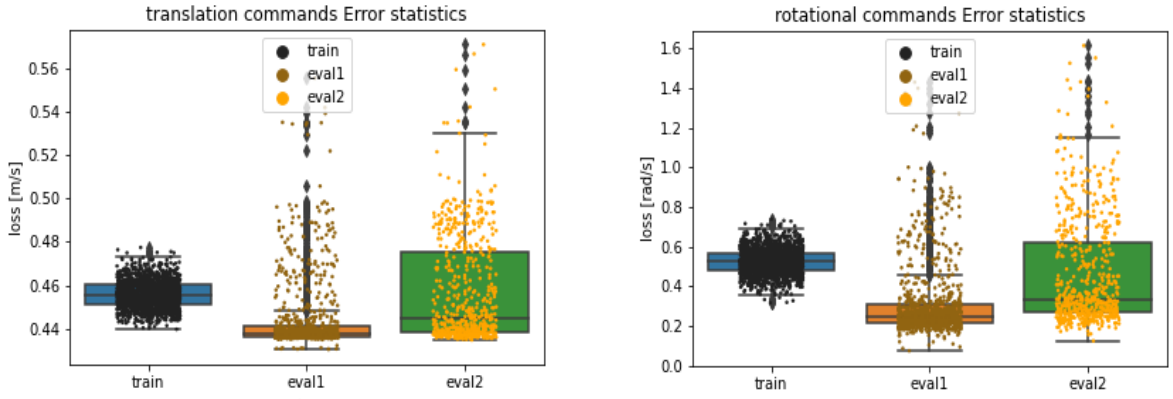

Figure 2.15 CNN-based mode: Error statistics of the frame-by-frame between ground truth and predicted steering commands of three different evaluation datasets

For the train losses distribution, it's clear that all of the train losses of both translation and rotational lie around the value that the model converged too, however the eval1 losses suffers from outliers specially for rotational losses. And this mainly due to the robot big rotational angles that that are larger than $\frac{\pi}{2} rad/s$ or less than - $\frac{\pi}{2} rad/s$. The same goes for the eval2 dataset, which has a higher variance in the rotational loss distribution, because it was collected to test difficult obstacle avoidance scenarios. Even though the median loss of the eval1 and eval2 datasets are less than the median of train losses, their mean is higher. And this happen because of the outliers caused by high steering angles. Which what the model failed to learn properly.

Figure 2.16 shows a comparison between the predicted and actual steering commands of 700 frames contained in eval2 dataset. The predicted translation velocities shown yellow (Figure 2.16 down) track the ground truth velocities commands very well, as expected from the evaluation losses. However, the predicted rotational velocities (Figure 2.16 up) are failing to track the actual one at values greater than 1.2 rad/s and lower than -1.2 rad/s. and this is a consequence for the unbalanced collected dataset, which has a relatively low fraction of data collected of high rotational commands.

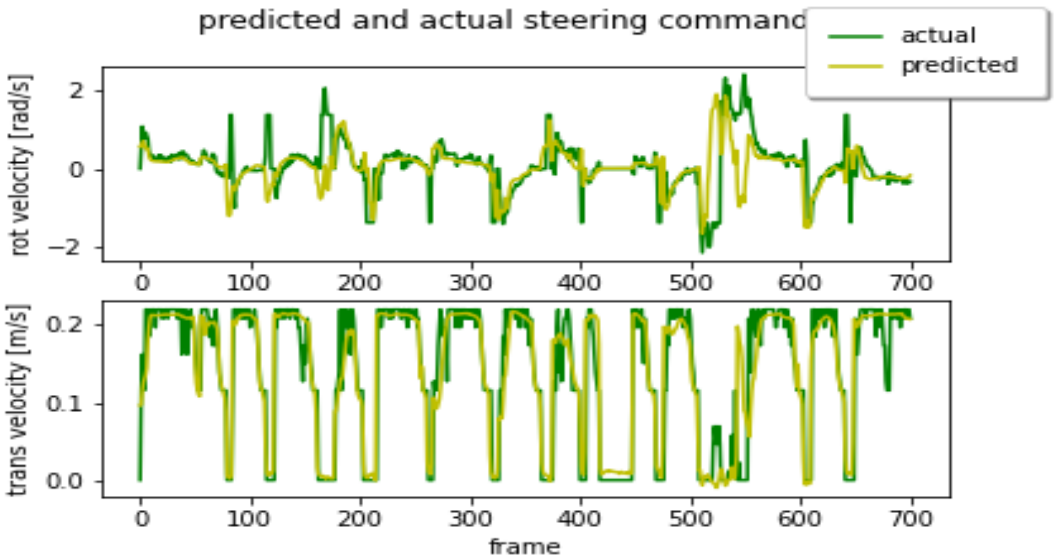

Figure 2.16 comparison between predicted and actual velocity commands for 700 frames in the eval2 dataset

Chapter 3: Improved Model with Temporal Fusion

The previously presented Convolutional neural network model computes the steering commands frame-by-frame. No memory is used in order to take into account previous inputs our outputs. Using previously extracted features to predict the next steering commands seems like a good idea. If the robot knows its previous states, it can generate a better prediction. Using the traditional neural network will likely to fail, since it has no explicit way to process a sequence of data. Instead, we will incorporate a new type of neural networks called **Recurrent Neural Networks** or **RNN.**

3.1 Related Work

Usually, Recurrent Neural Networks are used for Natural Language Processing (NLP), such as translation and speech recognition. Basically, it takes a sequence of data such as a sentence, and precited its translation. This approach has proven its effectiveness in NLP, however, in recent years it has been incorporated in several domain that may deals with end-to-end learning and data time series.

Another line of work has treated autonomous navigation as a visual prediction task in which future video frames are predicted on the basis of previous frames. Such as the work [10], which propose an approach to learn a driving simulator by combining generative adversarial neural network (GAN) and recurrent neural network (RNN). also, in recent advances, recurrent neural network modeling for sequential image data are also related to our work. such as The Long-term Recurrent Convolutional Network (LRCN) [11] model that investigates the use of deep visual features for sequence modeling tasks by applying a long short-term memory (LSTM) recurrent neural network on top of a convolutional neural network.

Applying a CNN on top of RNN is an approach that has been incorporated for self-driving cars steering control. Such as the work of [12],where they presented and end-to-end steering model for self-driving cars that is able to encode the spatiotemporal information from different scales for steering angle prediction. Also, the work proposed by the first-place team

winner of the Udacity self-driving-car navigation [13],where they used what's called Conv-LSTM layers that makes use of Temporal Visual Cues in the dataset. By considering the previous outputs generated by the system, the network is able to predict an optimal steering command. This approach has proven its superior performance over the end-to-end CNN models. While this previous mention researches are used for self-driving cars and relies on image data, we used lidar data as our visual input. Also, we incorporated a much simpler model in our second approach, which reserves the temporal information as well.

3.2 Approach

Starting from previously mentioned research, we enhanced our previous CNN model by incorporating a type of RNNs called **Long Short-Term Memory** Units or **LSTM**. Which is a more powerful version of RNN, that is heavily used by the NLP society. By doing So, the end-to-end model will have the ability to consider the previous successive features to generate a better steering command prediction.

Our goal is to predict the current steering command conditioned on the past and current extracted features. To accomplish this, we propose a novel architecture for time-series features prediction, which combine an LSTM temporal encoder with a convolutional visual encoder. At first, a lidar data encoder is used to learn the relevant visual representation of the environment, alongside with another encoder used for the target information. Afterward, a sequence of those encoded features is passed to a temporal network that is used to take advantage of the motion history information of the mobile robot.

The proposed model constitutes of two main sub-networks and a small memory, as depicted in Figure 3.1. The first sub-network is a is a CNN based feature extraction neural network that is responsible of extracting the lidar and target information features. in the other hand, the second sub-network is Recurrent Neural Networks that uses the LSTM units. Mainly, it is designed to learn through a data time sequence, which allows us to benefit from previous state of the robot to build a better model. So instead of using only the current extracted features, the RNN will used a sequence of successive features to predict a better steering command.

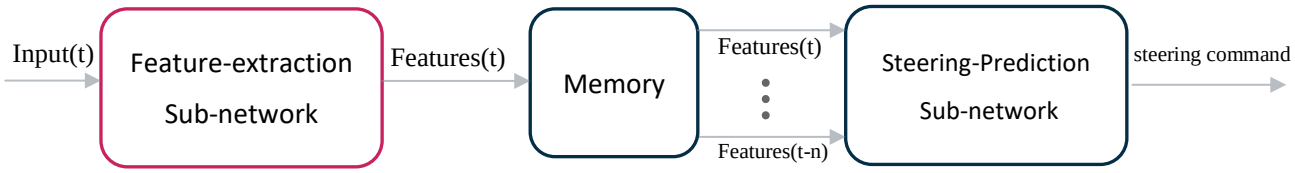

Figure 3.1 CNN-LSTM based Model

3.3 Model Structure

Unlike the previous network, this network is divided into two sub-networks as depicted in Figure 3.1. The first network is responsible for extracting features from lidar scan and target data. After that, those features are added to a set of features from previous time steps to be fed to the steering perdition sub-network.

3.3.1 Feature-Extraction Network

The features extraction network architecture is very similar to the previous main CNN architecture shown Figure 2.10. The convolutional layers remain the same except for the activation function where we used the Tanh instead of Leaky_ReLU. Also, no dropout probability associated for the convolutional layers. The last Max-Pool layer produces a $(1 \times 22 \times 256)$ Tensor, this tensor is to be flatten and propagated through a series of fully connected layers (dense layers) that has an output of dimension (1×256). Where the first column is reserved for batch size, the next is for features length.

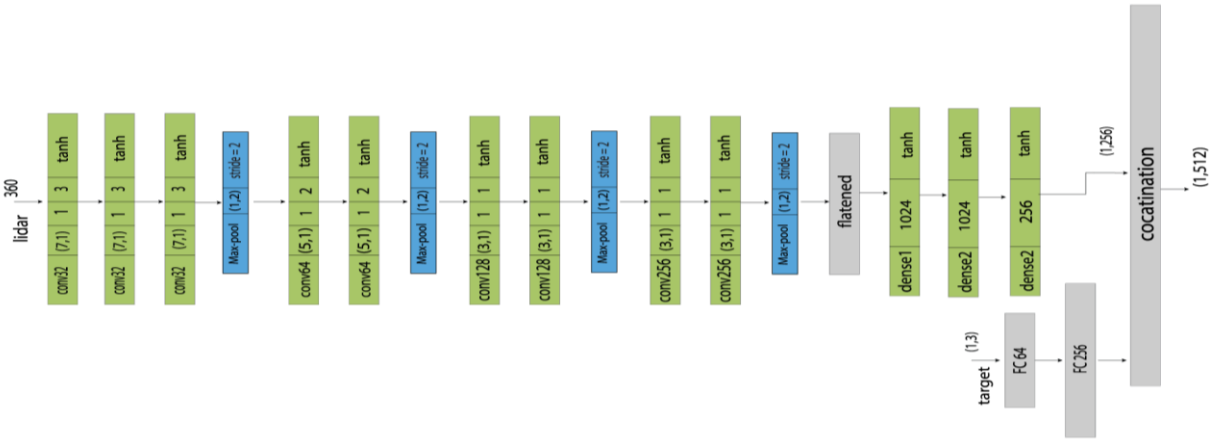

Figure 3.2 CNN_LSTM based model: features-extraction Network. (INPUTS: lidar data (1,360) and target location (1,3). outputs: features of dimension (1,512))

As shown in Figure 3.2. The (1×256) Tensor corresponds features extracted form lidar scan data. These features are then concatenated with the (1×256) target location up-sampled features to form a (1×512) data frame features. Later on, these features will be added to another set of features that had been propagated previously through the network.

3.3.2 Steering-Prediction Network

LSTM networks are an extension of recurrent neural networks (RNNs). mainly, it is a network that works on the present input by taking into consideration the previous output and storing in its memory. Unlike the CNN, where there is no memory associated with it. Which is a problem for sequential data, like text or time series. RNN addresses that issue by including a feedback loop which serves as a kind of memory. So, the past inputs to the model leave a footprint.

While the main advantage of RNNs is the consideration of past information, it turns out, if RNNs units considers future information, it can give a better result. Therefore, in our steering-prediction network, we used a bidirectional RNN which is constructed using LSTM units. In total, there are 16 LSTM units, half of them corresponds to forward pass while the other half corresponds to the backward pass. in Figure 3.3 each forward and backward LSTM unit are presented as one LSTM block, which makes a total of 8 LSTM bocks. This network takes as an input a time data sequence. These data sequence corresponds to a set of consecutive features extracted by the Feature-Extraction network. At each time step, the extract features will be stored in memory (RAM or GPU), and the last eight set features will be extracted memory and feed to the LSTM network.

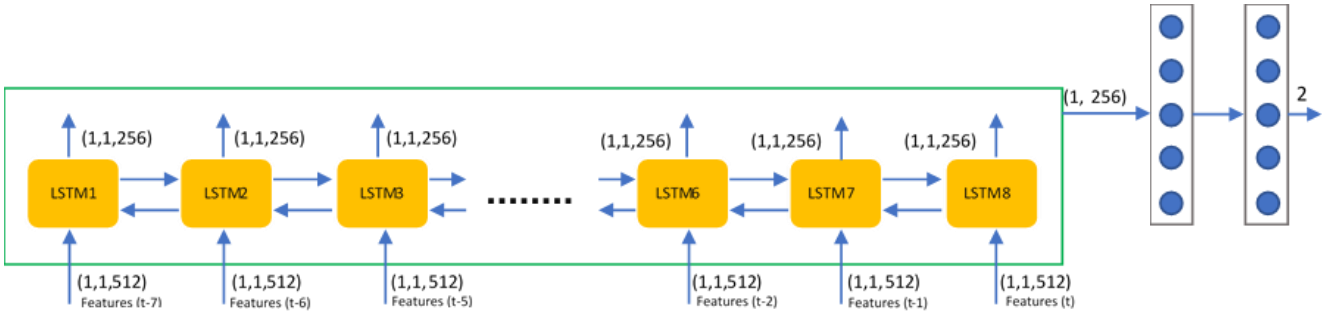

Figure 3.3 Steering-Prediction Network

Since this is a bidirectional RNN, each LSTM block has two LSTM units, one deals with the forward pass, and the other one deals with the backward pass. Which means, one provides

past information while the other one provides future information. As an example, The LSTM3 block, takes information from the past (LSTM2 forward unit), as well as information from present (Features (t- 5)), as well as information from future (LSTM4 backward unit). And by doing so, the LSTM3 unit can generate a better prediction.

In particular, we are interested in the prediction of the LSTM8 block, which deals with features from the current time step. So, by having the knowledge of previous time steps provided by the other LSTM blocks, and assuming that those LSTM block made a good prediction, the last LSTM8 block will have a good prediction as well.

The network generates eight Tensors of size (1,256) each. During deployment, only the last LSTM unit's tensor is used to predict the steering commands by propagating it through a series of dense layers. However, during training, all of the LSTM units will be propagated through the dense layers as will be explained in the next section.

3.4 Training details

Although, there is two networks in this model, both of them where trained in the same time. With same procedure as the previous model, the goal is to minimize the MSE loss function (2.6). as depicted in Figure 3.4. By setting the batch size to 128, and with Adam algorithm is used for back-propagation optimization with learning rate of 0.0001. The improved model is trained with 225,000 training data samples. And validated with 25,000 data samples. The training process took 34 epochs through the training set, and no significant improvements in the error rate occurred thereafter. After training, the average loss (mean-squared error) was 0.202 on the training dataset and it was 0.370 on the valid dataset.

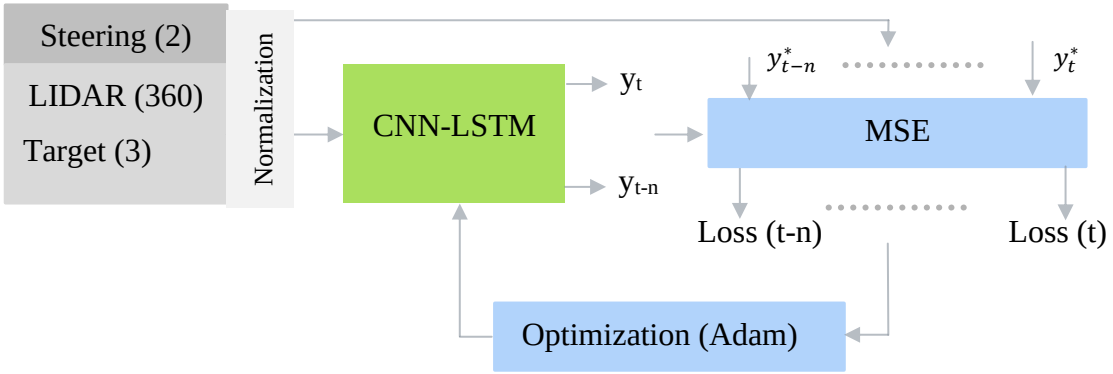

Figure 3.4 Training the CNN-LSTM neural network

Clearly, this approach is more complicated than the previous one. However, by incorporating the features of the previous time steps, the CNN_LSTM based model is able to learn an generalize better than the CNN based model. As shown in in Figure 3.5, the validation loss curve converges starting from the value “0.56” until it reaches the value 0.37. Note that, the valid loss at the first iteration of CNN-LSTM model is better than the best value “0.57” that the valid loss of CNN model converged to. And this shows the power of temporal fusion in navigation.

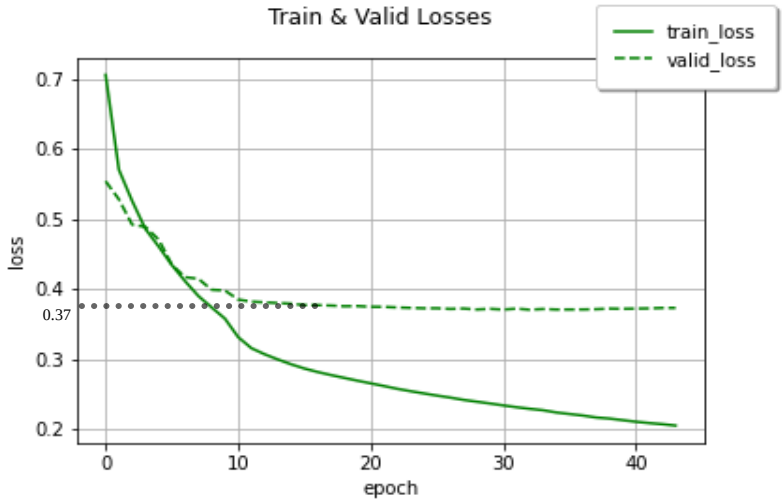

Figure 3.5 improved model Train and Eval Losses

Although, the CNN-based model found a difficulty in learning the rotational velocity commands, the validation loss of the CNN-LSTM based model ,shown in Figure 3.6 right, does converge to a much lower value than the previous model. And this proves that the improved model has managed to overcome that problem. And this mainly due to the incorporation of the previous state of LSTM units.

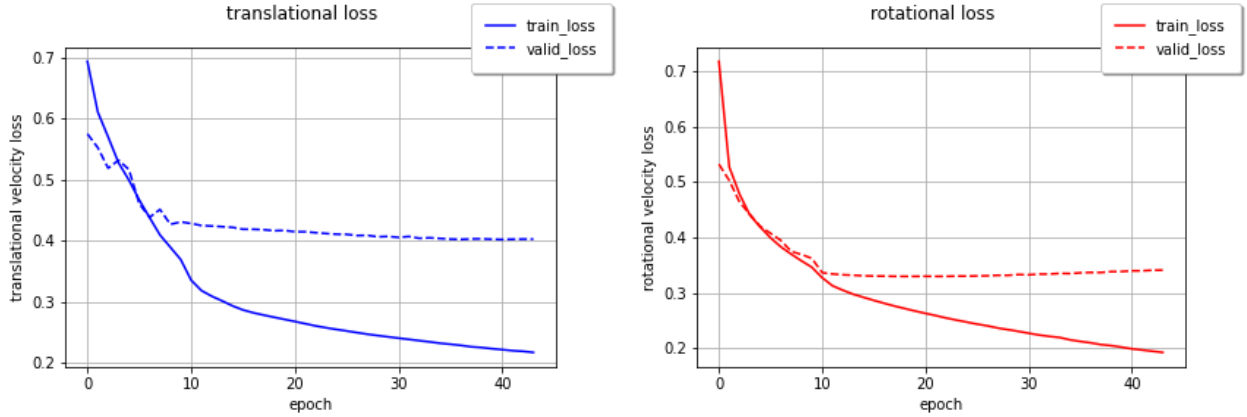

Figure 3.6 improved model: translation and rotational velocity commands losses

For the translation velocity commands loss, Figure 3.6 left, shows that, the valid loss had converged to a loss value equals to 0.4 which is approximately equals to the value that the CNN based model converged to. Therefore, both models are already doing a similar performance level in terms of translation velocity.

It is important to point to the process of features manipulation between the two sub-networks. The features extracted from the first network has dimension of (128×512), where the first index represent the batch size and the second index represents the features length. This batch of features are to be reshaped to obtain the successive data sequence that we need to feed it to the second network. and since we need a sequence of length equals to 8, that means in a batch size equals to 128 we can obtain 16 sequences. As a result, features extracted from the first sub-network will be reshaped to a tensor of shape (16×8×259) where each index comes as follow (min_batch, sequence, input).

By propagating the resultant tensor through the LSTM network, each element in the sequence is propagated through its corresponding LSTM block. And by doing so, each LSTM block generates a tensor of dimension (16×1×256) where 256 corresponds to the number of hidden states in the LSTM block. Therefore, the whole LSTM network generates a tensor for (16×8×256). In order to make each LSTM block generates its own steering prediction, the generated tensor is reshaped to (128×256). Which will be propagated through two dense layers, the final dense layer will have output size of (128×2). This way, each LSTM block is trained to generate its own steering command. However, during deployment we will propagate only the output of the LSTM8 block through the dense layers

3.5 Evaluation

Clearly, the translation and rotational frame-by-frame losses, shown in Figure 3.7, have a lower variance than that of the losses of the previous model. Also, the eval1 and eval2 rotational losses have been compressed to a lower range, and this shows the improved model is able to make less error in predicting the rotational velocity.

In general, the second model helped in reducing the outlier with high values, which cause the mean of validation losses of the first model to be relatively higher. However, the improvement of the second approach to our end-to-end model does not hide the fact that, in order for the validation losses to be closes to zero, more data is needed.

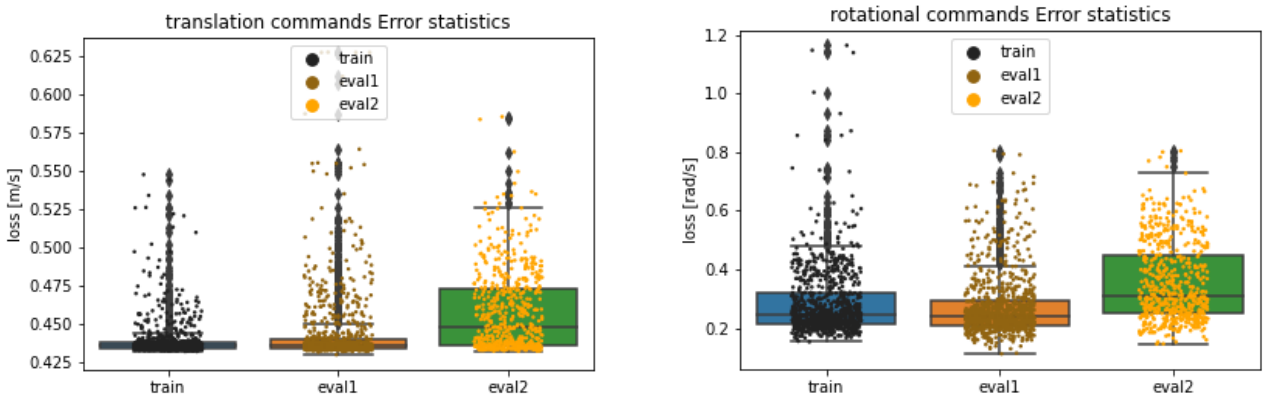

Figure 3.7 Improved model: error statistics of the frame-by-frame between ground truth and predicted steering commands of three different evaluation datasets

Figure 3.8 shows the plot of the predicted and actual rotational and translation velocities. Clearly, the predicted rotational velocity (Figure 3.8 up in yellow) is tracking the actual one, which is drawn in red, very well. In contrast to the previous model, incorporating temporal features helped a lot with the rotational velocity commands. However, the translation velocity plot Figure 3.8 down) shows some misprediction at some frames. For example, at frame 420 till frame 450, the network prediction indicates that the robot needs to move, while in reality it should stop. Also, at the start, the network shows some inconsistency between predicted and actual velocity commands, and this happened because the network needs eight consecutive features, which should be extracted while the robot is moving. However, at the start the robot is stationary, which means the extracted features corresponds to the same location. Therefore, the prediction will be not good until the eight features are collected while the robot starts to move.

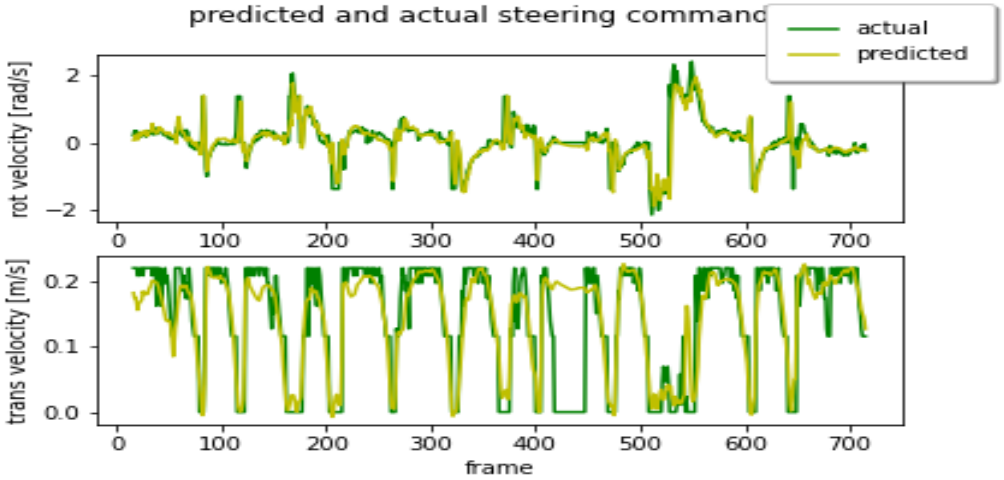

Figure 3.8 improved model: comparison between predicted and actual velocity commands for 700 frames in the eval2 dataset

Chapter 4: Experiment and Results

In this chapter, we present the details of our experiments in section 4.1, which includes software used for building the end-end systems as well as deploying them. The experimental results are given in section 4.2, and they are discussed in section 4.3

4.1 Experiment Setup

4.1.1 Software Platform

First of all, the whole system was developed and tested on ubuntu 18.04 operating system, which runs onboard a computer that is equipped with i5 6300HQ processor, 8GB of RAM and a Graphics processing unit GTX 960M.

The parts that concern data collection and model deployment were developed using both C++ and Python programing languages across the Robot Operating System (ROS) framework [14]. Likewise, the training process was done using Pytorch framework [15].

4.1.2 Software Architecture | Data Collection

Data collection was done through several steps. First of all, while the Turtlebot was being driven in the simulation environment, a built-in rosbag package used to collect data from several ros topics, each of which has its own frequency. After the data has been collected, a C++ based package was used to synchronize the data based on their time steps. Later on, this data was parsed to CSV format using a CSV_Parser python-based package. This whole process is shown in diagram of Figure 4.1.

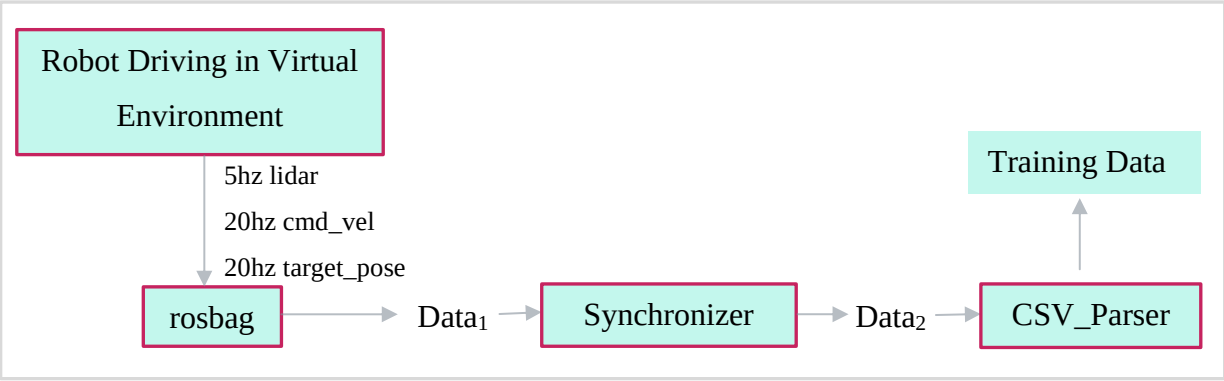

Figure 4.1 Data Collection| Software Architecture

Basically, the Synchronizer package has two main tasks to do; the first one is transforming the reference frame of all the incoming data to the robot body reference frame. The second task is synchronizing that data based on their time step in each frame's header, therefore after synchronization, data will have the same number of samples for each topic, however, the data size will much less than the original one.

The csv parser package was construct to convert data from bag format to csv format. so, the package takes as input a bag file and parse that bag file to a set of csv files. Consequently, we will have a three csv files, the first on corresponds to the target locations where each row contains the x, y position and the heading θ. The second file corresponds to steering commands $[v, \omega]$. The last csv file is the lidar data file where each raw has a size of 360 that corresponds to the point clouds range.

4.1.3 Software Architecture | Deployment

The trained model was deployed using ROS and Pytorch. The input data was coming at different frequency, therefore, a Synchronizer package similar to the one used before in data collection is used to provide the model with a synchronized data at frame rate of 5hz. as shown in Figure 4.2. The synchronization process was running on CPU while the navigation model was running on GPU. Using GPU computation ability, both models where running in real time. the CNN model was able provide prediction at an average rate of 100hz, while the CNN-LSTM based model was running at an average rate of 50hz. even though both models where running at higher frequency, the input data was limited to 5hz, and this due to limited LIDAR scanning frequency. Therefore, the whole end-to-end model was giving steering commands at frequency of 5hz.

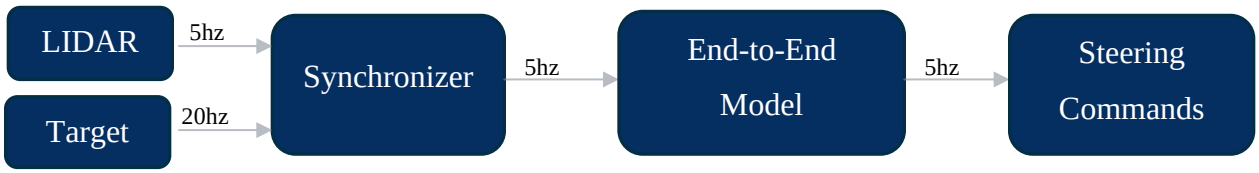

Figure 4.2 Software Architecture | Deployment

4.2 Experiment Results

Figure 4.3 and Figure 4.4 show two navigation scenarios, in each one of them the robot is shown in a red square and is trying to navigate to the goal shown in green square. both of these scenarios are part of the eval2 dataset, which has not seen before by any of the trained networks. In order to evaluate the performance of both networks, we compare the actual steering commands with the ones predicted by the first and second networks. the comparison was done throughout two navigation scenarios, and the results of each scenario are given in Table 1 and Table 2 respectively. These results were recorded approximately at each one second, given that navigation model is publishing velocity commands at a rate of 5Hz.

As shown in Table 1, the predicted translation velocity values for both networks are very close to the actual predicted values. however, the second network predicts rotational velocities that is closer to the actual one than the prediction of the first network which shows some margin between the actual and the predicted values. For example, at the situation shown in Figure 4.3 C1, the robot is steering away from the obstacle, in which the actual rotational velocity is $\omega = -0.624\, rad/s$, however, the CNN network predicted $\omega = -0.432\ rad/s$, and the LSTM based network predicted $\omega = -0.557\, rad/s$ which is a closer to the actual value.

Table 1 List of Actual and Predicted Steering commands of navigation scenario in ***Figure 4.3***

Navigation Scenario 1	Actual Steering (DWA)	Predicted Steering First Model (CNN)	Predicted Steering Second Model (LSTM)
A1	$v = 0.127\ m/s$ $\omega = -1.303\, rad/s$	$v = 0.191\ m/s$ $\omega = -0.317\ rad/s$	$v = 0.186\ m/s$ $\omega = -0.685\, rad/s$
B1	$v = 0.220\ m/s$ $\omega = -0.923\ rad/s$	$v = 0.19\ m/s$ $\omega = -0.409\ rad/s$	$v = 0.205\, m/s$ $\omega = -0.557\, rad/s$
C1	$v = 0.208\ m/s$ $\omega = -0.624\, rad/s$	$v = 0.200\ m/s$ $\omega = -0.432\ rad/s$	$v = 0.201\ m/s$ $\omega = -0.504\, rad/s$
D1	$v = 0.22\ m/s$ $\omega = 0.201\, rad/s$	$v = 0.210\ m/s$ $\omega = 0.139\ rad/s$	$v = 0.207\ m/s$ $\omega = 0.143\, rad/s$
E1	$v = 0.22\ m/s$ $\omega = 0.279\ rad/s$	$v = 0.210\ m/s$ $\omega = 0.215\ rad/s$	$v = 0.213\ m/s$ $\omega = 0.258\, rad/s$
F1	$v = 0.22\ m/s$ $\omega = 0.073\ rad/s$	$v = 0.20\ m/s$ $\omega = 0.008\ rad/s$	$v = 0.205\ m/s$ $\omega = 0.047\, rad/s$

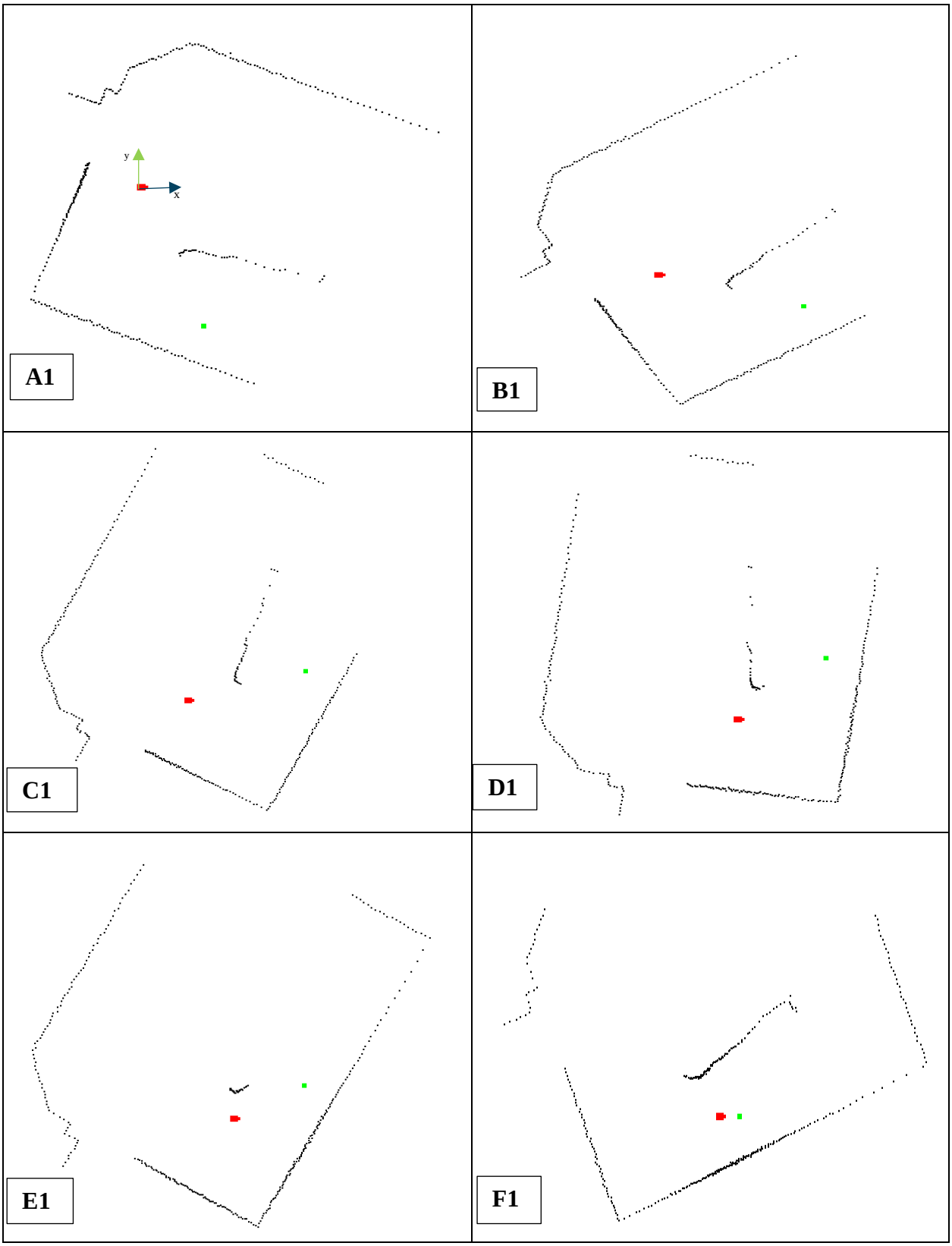

Figure 4.3 First Navigation Scenario of Robot (Red) navigating toward the goal (green)

The results of the second navigation scenarios of Figure 4.4 are shown in Table 2. the values predicted by the second network are closer to the actual one than the those predicted by the first network. However, for the situation in Figure 4.4 A2, the first network is predicting the wrong rotational angle. the robot is steering away from the obstacle as indicated by the actual value $\omega = 0.296\, rad/s$, however the LSTM-based network is generating a negative steering angle $\omega = -0.172\, rad/s$, as if it is trying to avoid the obstacle from the other direction. Although, once the robot advances a little bit (see B2), the LSTM-based model predicts the same values as the actual steering values.

Before the robot starts to navigate, the robot was stationary. And this means the LSTM-CNN based network has no past information, therefore at situation A2, the temporal information that the CNN-LSTM model was using in order to predict the first steering command corresponds to a sequence of features that are extracted from the same position, since the robot was not moving before. And this may explain why the CNN-LSTM based network predicted completely different value than the actual one. In the other hand, the CNN-based network is steel predicting a steering value with some margins between it and the actual once.

Table 2 List of Actual and Predicted Steering commands of the Second navigation scenario in ***Figure 4.4***

Navigation Scenario 2	Actual Steering	Predicted Steering First Model (CNN)	Predicted Steering Second Model (LSTM)
A2	$v = 0.208\ m/s$ $\omega = 0.296\, rad/s$	$v = 0.205\ m/s$ $\omega = 0.168\, rad/s$	$v = 0.208\ m/s$ $\omega = -0.172\, rad/s$
B2	$v = 0.220\ m/s$ $\omega = -0.052\ rad/s$	$v = 0.215\ m/s$ $\omega = -0.113\ rad/s$	$v = 0.215\, m/s$ $\omega = -0.054\, rad/s$
C2	$v = 0.220\ m/s$ $\omega = -0.173\, rad/s$	$v = 0.216\ m/s$ $\omega = -0.218\, rad/s$	$v = 0.210\ m/s$ $\omega = -0.173\, rad/s$
D2	$v = 0.22\ m/s$ $\omega = -0.327\, rad/s$	$v = 0.216\ m/s$ $\omega = -0.241\, rad/s$	$v = 0.213\, m/s$ $\omega = -0.236\, rad/s$
E2	$v = 0.220\ m/s$ $\omega = -0.180\ rad/s$	$v = 0.214\ m/s$ $\omega = -0.191\, rad/s$	$v = 0.219\ m/s$ $\omega = -0.187\, rad/s$
F2	$v = 0.196\ m/s$ $\omega = -0.067\ rad/s$	$v = 0.206\ m/s$ $\omega = -0.043\, rad/s$	$v = 0.197\ m/s$ $\omega = -0.025\, rad/s$

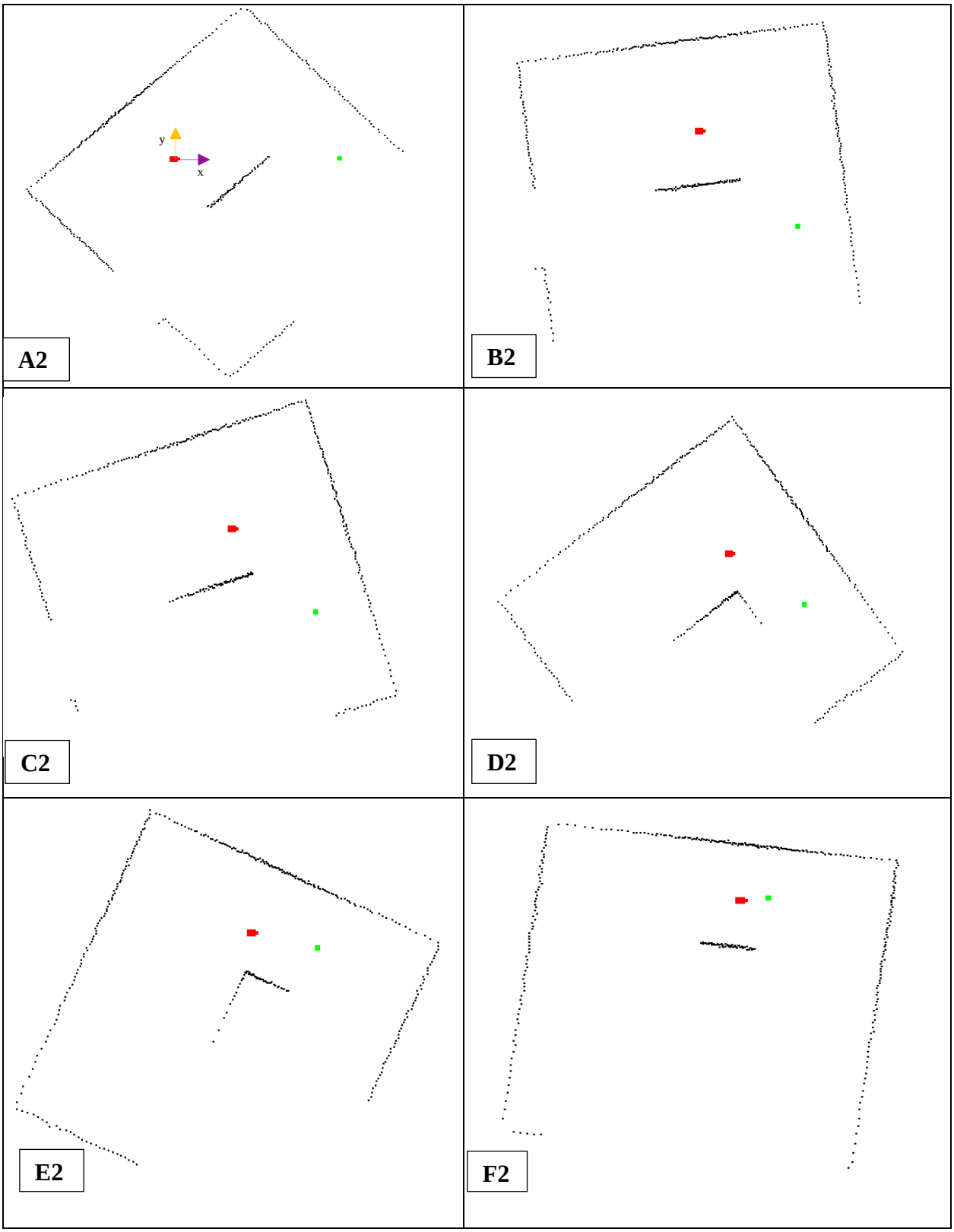

Figure 4.4 Second Navigation Scenario of the Robot (Red) navigating toward the goal (green)

4.3 Discussion

This work showed successfully that a mobile robot navigation policy can be learned from an expert operator making use of neural network learning ability of learning a complex end-to-end mapping functions from raw sensor data to steering commands. two different neural network architectures have been trained and showed their abilities to transfer the gained knowledge from training environments to unseen environments.

Out of the two trained networks, the CNN based model relies only on the current time input frame, whereas the CNN-LSTM based model uses a sequence of the last eight input frames to predict the next steering commands. In terms of advantages and disadvantages of each model, the first model has a 2X better query time than the second model. Also, the two networks performance is very close in terms of the translational velocity prediction. However, the second network predictions of rotational velocities are relatively better, and this because the second network makes use of the previous hidden states of the LSTM network.

The results of the first navigation scenario shown in Table 1 lists the steering command predictions from different time steps. Obviously, the results of the CNN-LSTM based are closer to the base line results than the those of the CNN model. However, this does not mean that the first model will hit an obstacle, since those results are taken from frames which are one second apart. Means, in a certain time step, the CNN model may predict a low rotational angle commands, and in the next time step it may keep up with the situation and predict a higher one.

As shown in Table 2, The CNN-LSTM based model clearly does a better job in predicting the rotational velocity. However, there is certain situations where it may fail to predict reasonable commands. The first one occurs when the robot starts to navigate, it generates steering commands that may be not good enough to avoid an obstacle if it was very close to it. The reason for this comes from the property of the previous LSTM hidden states, which have big influence on the prediction of the next steering commands. Giving all of these states correspond to features extracted from inputs of a stationary robot. This means the model will get a sequence of eight features that are the same, in which the trained model does not have knowledge of how to deal with such situation. Therefore, this problem may be solved by including such scenarios in the training data and retraining the model.

Another issue happens to both models, it usually occurs whenever the robot approaches the goal, both models reduce forward velocity to be close to zero and aim to align the robot with goal heading. However, the robot will never get perfect alignment with goal's heading, and this may cause the robot to start spinning about the target. therefore, a tolerance value has been set to avoid such scenario.

An important issue needs to be addressed for this navigation system, given the network was trained with target location in the vision range of the robot, the system was not trained to explore the environment like a maze and search for a possible way to the target location. Although we compare our approach to the DWA which is a map-based motion planner, we are fully aware that it cannot completely replace a map-based path planner. If the environment becomes more complex, the navigation system is still limited to be a local motion planner that relies on a global path planner to provide targets.

There is a certain favorite property of using the end-ot-end learning approach, which is the fact that it can be used for both dynamic and static environment. Specially the second model, which has more sense in tracking the motion of other dynamic objects in the environments. Even though, the data used to train our model corresponds to navigation in static environment, which means our model cannot navigate in dynamic environment. If another data was collected from environments which contains dynamic objects, this approach is steel a solution to this problem as well. However, in traditional approach, dynamic objects have to be detected and tracked by separate module, and it has to provide locations of each dynamic object of the environment, giving the complexity increases with increasing in the number of dynamic objects in the environment. The end-to-end model does not have to deal with all this problems as it combines them to one problem which is considered as the main advantage of this approach.

Chapter 5: Conclusion

In this work, we presented a data-driven end-to-end navigation approach for autonomous mobile robots. This approach relies on the lidar scan to control a mobile robot through an indoor environment seeking a relative target location. the system uses the ability of neural networks to learn a complex function that maps the input data to a steering commands of a differential drive mobile robot.

This report presented two distinct end-to-end models that are able to learn navigation strategies from an expert operator and generalize this knowledge to previously unseen environments. At first, a CNN based model was presented in chapter 2, where it was trained to get a current input frame and output steering commands. This model has shown it ability to navigate in previously unseen environments and avoid obstacles while reaching the target. however, it showed some difficulty in producing high angular velocity commands when it encounters very close obstacles. which lead us to propose a novel model that fuses the temporal information to produce a better steering command prediction.

The CNN-LSTM based approach was presented in chapter 3, and it showed the ability to predict a higher angular velocity in difficult obstacle avoidance situations. this model uses an LSTM network to fuse all past and current states into a single state which allows the model to predict a better and smoother steering commands. As a result, the obstacle avoidance ability has been improved. Also, compared to the first model, the second model produces smoother steering commands, that allows for smooth navigation experience.

In the future, our main focus will be on making the robot navigates in more complex environment, meaning teaching the robot how to explore the same why humans explore to reach a goal location. There is much existing research being done in this area that tackles this specific problem, such as the work proposed by Piotr Mirowski [16]. Which provides an approach to the problem of navigation in complex environments using a deep Reinforcement Learning solution. The approach is tested in visually rich simulated 3-D game-like maze environments and provides an analysis of results that compare favorably to human behavior for the same mazes.

References

[1] S. Roland, N. Illah Reza and S. Davide, Introduction to Autonomous Mobile Robots, The MIT Press, 2011.

[2] D. A. Pomerleau, "ALVINN: An Autonomous Land Vehicle In a Neural Network," 1989.

[3] L. Yann, M. Urs, B. Jan, C. Eric and F. Beat, "Offroad obstacle avoidance through end-to-end learning," *in Advances in Neural Information Processing Systems,*, pp. 739-746, 2005.

[4] B. Mariusz, D. T. Davide, D. Daniel, F. Bernhard, F. Beat, G. Prasoon, D. J. Lawrence, M. Mathew, M. Urs, Z. Jiakai, Z. Xin and Z. K. Z. Jake, "End to End Learning for Self-Driving Cars," 2016.

[5] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell and M. Hebert, "Learning Monocular Reactive UAV Control in Cluttered Natural Environments," *IEEE International Conference on Robotics and Automation (ICRA),* p. 1765—1772, 2013.

[6] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart and C. Cadena, "From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots," *IEEE International Conference on Robotics Automation (ICRA).,* p. 527–1533, 2017.

[7] E. Ackerman, "TurtleBot 3: Smaller, Cheaper, and Modular," *IEEE Spectrum,* 2016.

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

[9] J. B. Diederik P. Kingma, "Adam: A Method for Stochastic Optimization," *3rd International Conference for Learning Representations,* 2015.

[10] E. Santana and G. Hotz, "Learning a Driving Simulator," 201.

[11] D. Jeff, A. H. Lisa, G. Sergio, R. Marcus, V. Subhashini, S. Kate and D. Trevor, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description," 2015.

[12] T. Wu, A. Luo, R. Huang, H. Cheng and Y. Zhao, "End-to-End Driving Model for Steering Control of Autonomous Vehicles with Future Spatiotemporal Features," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* p. 950—955, 2019.

[13] Y. M. Lu Chi, "Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues," *arXiv,* 2017.

[14] W. G. S. A. I. Laboratory, "Robotic Operating System(ROS) melodic," 2018.

[15] P. Adam, G. Sam, C. Soumith, C. Gregory, Y. Edward, D. Zachary, L. Zeming, D. Alban, A. Luca and L. Adam, "Automatic differentiation in PyTorch," *NIPS 2017 Workshop Autodiff Submission,* 2017.

[16] M. Piotr, P. Razvan, V. Fabio, S. Hubert, J. B. Andrew, B. Andrea, D. Misha, G. Ross, S. Laurent, K. Koray, K. Dharshan and H. Raia, "Learning to Navigate in Complex Environments," *The International Conference on Learning Representations (ICLR),* 2017.

[17] A. H. N Koenig, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *Intelligent Robots and Systems,* 2004.

[18] E. Ackerman, "TurtleBot 3: Smaller, Cheaper, and Modular," *IEEE Spectrum,* 2016.