ABSTRACT

BHANDARI, LAKSHAY RAKESHKUMAR. Navigation of Autonomous Differential Drive Robot using End-to-End Learning. (Under the direction of Edward Grant).

Autonomous driving is and has been one of the most prevailing robotics research topics in recent years. Having said that, it draws attention that self driving vehicles demand a tremendous amount of research from a variety of topics in engineering, mathematics and science. Challenges remain in the fields of perception, localization and control. A basic perception application that uses a camera feed requires an exhaustive pipeline which includes data acquisition, image processing for feature extraction followed by the application of an estimation model from classical machine learning, or deep learning, to identify objects in the image using extracted features. Following the feature extraction phase, an algorithm for object localization within the scene is applied; in order to generate instantaneous motion control.

The research here follows on from research originally conducted by NVIDIA into end-to-end learning for autonomous vehicles. The goal of the research here is focused on the reduction of the pipeline size and on the effort required to implement multiple algorithms for a range of tasks, e.g., tasks including feature extraction, object detection and object localization, for an autonomous driving Wheelchair. Using just RGB images as the primary perception input signal, the developed system generates instantaneous driving commands for this Wheelchair.

Experiments with the above approach, embedded within a single trained Convolutional Neural Network, can handle an entire perception pipeline and thereby reduce the burden on both the designer and the hardware used.

© Copyright 2019 by Lakshay Rakeshkumar Bhandari

All Rights Reserved

Navigation of Autonomous Differential Drive Robot using End-to-End Learning

by
Lakshay Rakeshkumar Bhandari

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2019

APPROVED BY:

Andrew Rindos	John Muth

Edward Grant
Chair of Advisory Committee

DEDICATION

To mumma, papa and my sisters who have been tremendously instrumental in my journey so far.

BIOGRAPHY

The author hails from Surat, a city in Gujarat, India. Prior to joining the ECE graduate program at NC State University, he worked as a Controls Engineer at Reliance Industries Limited, India, following the completion of his Bachelor of Technology in Instrumentation and Control Engineering from Nirma University in 2015.

ACKNOWLEDGEMENTS

The work performed in this thesis over the course of two semesters has been possible due to the indispensable support provided by my advisor, Dr. Edward Grant for fuelling my enthusiasm in research in a highly budding and interesting field. All my learning in the duration of this thesis has greatly consolidated my understanding of the concepts employed in this work. My senior colleague, Hamed Mohammadbagherpoor has been very generous and kind throughout, helping me whenever I was stuck at any stage.

My fellow researcher, Abhayjeet Singh Juneja, has been my companion in the endeavour since the time we started working on the Wheelchair and experimenting with it for integration of AI into the navigation control system. We have spent innumerable hours developing the system that was used in this work, learning a lot from each other. I would also like to thank Dr. Andrew Rindos and Dr. John Muth for being members of my committee. I am grateful to the people who have extended their help to collect hours worth of training data. I can recall making them stand or walk in front of the wheelchair while I would operate the wheelchair.

In addition, I cannot thank enough the people outside my academic bubble who have motivated me to work hard each day. My mother, who has been always there to push me and channelize my passion in right directions. My father and my sisters, for their constant love and guidance. I am also grateful to my friends and roommates, who have helped me in my hard times, both in academics and my personal life. All the support has made me feel at home and made my passion in engineering and science even stronger and fuelled my gusto to come up with new and efficient technologies to contribute to the society.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **vii**

LIST OF FIGURES . . . . . **viii**

Chapter 1 INTRODUCTION **1**
1.1 Thesis statement 3
1.2 Contributions 3
1.3 Organization 5

Chapter 2 BACKGROUND **6**
2.1 A Self-Driving Application 7
2.1.1 Perception 8
2.1.2 Localization 17
2.1.3 Path Planning and Control 19

Chapter 3 RELATED WORK AND MOTIVATION **21**
3.1 End-to-End Learning by NVIDIA 21
3.2 Precursor to this Work 24
3.2.1 Overview of the Test-Bench 24
3.2.2 AUTONOMOUS NAVIGATION 25
3.2.3 CHOICE OF PERCEPTION SENSOR FOR END-TO-END LEARNING 33

Chapter 4 SYSTEM DEVELOPMENT AND HYPOTHESIS **35**
4.1 Sensors: Perception 38
4.1.1 Kinect 38
4.1.2 Webcam 38
4.2 Software and Hardware Architecture 40
4.2.1 ROS Architecture 42
4.2.2 Hardware pipeline 44
4.3 The CNN 45
4.3.1 Network Architecture 45
4.3.2 Data Collection and Training 45
4.3.3 Experimentation 47
4.4 Motion Control 49
4.5 TussleNet : Not a Net 50

Chapter 5 EVALUATION **52**
5.1 Results 53

Chapter 6 CONCLUSION **61**

Chapter 7 FUTURE SCOPE **63**
7.1 Using RNN 64
7.2 Integration with IoT 64

BIBLIOGRAPHY **66**

LIST OF TABLES

Table 5.1 Comparison between end-to-end and other approach 53

LIST OF FIGURES

Figure 3.1 Structure of Kinect node in ROS 25
Figure 3.2 Object Detection in RGB image and corresponding Depth image . . . 28
Figure 3.3 Error calculation for Depth Precision Control 30
Figure 3.4 Orientation of LiDAR sensor on Wheelchair (Top View) 30
Figure 3.5 ROS node for LiDAR and Proximity Mapping algorithm 33

Figure 4.1 Wheelchair system 37
Figure 4.2 Installation of webcam on Wheelchair 39
Figure 4.3 LiDAR scan of environment same as captured in Fig. 4.4 40
Figure 4.4 (From top) Webcam capture, Kinect capture 41
Figure 4.5 ROS architecture for training the CNN 43
Figure 4.6 ROS architecture in deployed system 44
Figure 4.7 Hardware pipeline 45
Figure 4.8 CNN architecture for Regression Model 48

Figure 5.1 Summary for trained model 54
Figure 5.3 Performance scenario 2 54
Figure 5.2 Performance scenario 1 55
Figure 5.4 Performance scenario 3 55
Figure 5.5 Performance scenario 4 56
Figure 5.6 Performance scenario 5 56
Figure 5.7 A feature map in the middle of CNN 57
Figure 5.8 Feature maps following conv2_2d layer (indoor) 58
Figure 5.9 Feature maps following conv2_2d layer (outdoor) 59
Figure 5.10 Feature maps following conv5_2d layer 59
Figure 5.11 Training loss for CNN 60

CHAPTER

1

INTRODUCTION

The rise of Artificial Intelligence in recent years has fulled various innovations in the fields which have rather proved highly useful in human assistance in as simple tasks as reading to as complex as driving. With the powerful hardware available to cater to the ever growing computational needs of state of the art AI like Computer Vision, Autonomous Driving has become one of the most sought after and researched topics, resulting in unprecedented advancements in the field. It is expected that by the year 2020, the number of self-driving cars on US roads will sore to as high as 10 million. In addition, with the advent of Internet of Things, the number of connected smart vehicles is expected to grow to 250 million [Woo18].

With modern cars going electrical and being shipped with more autonomous features,

the technology primarily aims to eradicate traffic incidents which are caused due to reasons like distraction of the drivers while driving. In 2016, 84% of the car crashes recorded within the US were rooted in distracted drivers. Other major contributor to car crashes found was impaired drivers [Sta18], [Dav15]. Self-driving cars hold the potential to reduce the number of accidents. On the contrary, it cannot be stated that with today's self-driving technology the accidents can be absolutely avoided. There have been cases which have put a halt to deployment of self-driving cars on the road [The18], [Gri18].

Despite a few setbacks, there are a number of companies which are working to develop state of the art technologies which can aid the self-driving cars [Fri17], both including start-ups [Dix18] and well established giants [CBI18].

There are a number of steps involved for a single decision to be made by the autonomous vehicle on the road which involve data acquisition and processing, perception, localization and motion planning. Perception is usually done using sensors like Cameras, LiDARs, Ultrasonic, Radars either individually or with their signals fused together. With cameras, where the principal signal is an image, a typical pipeline involves image processing, classification techniques, object detection and localization within the scene and generating a control action (instantaneous velocity or driving angle). This type of an approach, which is often used for self-driving applications (e.g., HoG feature generation for detecting pedestrians using Support Vector Machines [DT05a], obstacle detection in an image using a Neural Network [How17] followed by bounding box estimation for each detected obstacle [Liu16a], [RF17a], e.t.c.) not only pushes hardware in the heavy lifting region, but also increases the complexity of the system with multiple algorithms to be interfaced with one another.

In this work, an approach previously used by NVIDIA [Boj16] is extended to autonomous driving for Differential Wheel Drive robots in an indoor environment, with further modifications made at the output of the system. With a single input image of the scene, the

system can generate either of instantaneous velocity for robot wheels or driving angle (angle with respect to the instantaneous motion) in contrast with the model proposed by NVIDIA where motion prediction is given in the form of the turning radius. Furthermore, an interactive pipeline is created which allows the user to take control in cases where the autonomous mode reflects inaccurate predictions. As a result, it could be seen the performance in the indoor environment was comparable with the previous implementation for self-driving cars on the roads. Also, it could be inferred that the outcome could be achieved with comparable performance with predictions being in the form of instantaneous velocity and driving angles. Furthermore, generating driving angle predictions gives the liberty to use system with regression or discrete output meaning either of a regression model or a classification model can be used.

1.1 Thesis statement

"Current trends in perception for autonomous differential wheel drive robots use a complex pipeline for navigation in the indoor environment, which can be simplified by leveraging the end-to-end approach to encompass the system into a single neural network."

1.2 Contributions

This thesis delves in a unique approach for navigation for the robots involving differential drive instead of a steering wheel as in cars. It aims to reduce the pipeline size and complexity by mainly avoiding the need for different algorithms for obstacle detection and localization. It is noted that the neural network used for object detection or classification can be modified by changing the output and the type of training data. More specifically, the neural network

can be modified to map the input image to driving commands instead of detecting the presence of an object or type of an object. Furthermore, the need to use an additional algorithm to find the location of the detected objects in the image can be prevented as well. Since the proposed system produces motion prediction based on the input scene image, detection of objects and their location is done inherently and thus using the obstacle detection or classification becomes redundant.

This work makes the following contributions:

- It is seen that the system developed for the differential drive robots for the environment with stated approach yields comparable results to the previous implementation for self-driving cars on roads [Boj16].
- We modify the system to produce the motion prediction based on the input image as either driving angles or instantaneous velocity directly. With output as the instantaneous velocity, the need to convert driving directions to velocity is also avoided.
- A separate pipeline is then built to train the neural network on the data where the predictions are inaccurate. This can help the user to train the network while in autonomous mode. We call this idea, rather than the CNN itself, the TussleNet as it aims to capture the tussle between the user and the CNN.

As stated above, this is not the first work in the field that uses this approach for autonomous driving application. We experiment with this tried approach to extend its application in the indoor environment and differential drive robots. Previous notable works have been published in the similar light in the past stressing on the end-to-end approach [Boj16], [Baj09].

1.3 Organization

The remainder of this thesis is structured as follows.

Chapter 2 provides a background on the technologies used in this work, and explains a general pipeline used by many of the autonomous driving applications, especially in the perception system. Chapter 3 provides the previous iteration of this work, along with the other relevant work which have used a similar approach. In Chapter 4, we provide our approach on the perception for autonomous differential drive robots. The hypothesis is explained by providing a use case for a Wheelchair system which serves as a test bench for this work. In Chapter 5 we summarize the performance evaluation of the system and explain how the this approach can be used to reduce the size of perception pipeline. Finally, in chapter 6, we conclude this work, and briefly touching upon its future scope in chapter 7.

CHAPTER

2

BACKGROUND

This chapter will define a general methodology that can be used in any self-driving application, be it for autonomous cars or differential drive robots; such as a wheelchair. Building on that, the algorithms involved, such as for obstacle detection and localization, will be explored in detail using classical machine learning techniques and also the more recent and powerful deep learning techniques. There will be exploration into how these components can be integrated to build a self-driving application. In addition, semantic segmentation will be explored, as will instance segmentation and other benchmark works which laid the foundation for computer vision techniques. These approaches contribute hugely to self-driving applications.

2.1 A Self-Driving Application

Self-driving technology is already functioning and still evolving and showing improved performance over previous iterations. As self-driving cars are meant to imitate human behaviour, any such application is composed broadly on the following components:

- Data Acquisition and Processing: this includes sensors information being received by the system and the processing of the information in order to be understood by the algorithms downstream. This part may or may not involve sensor fusion.
- Perception: this includes building a map of the surroundings based on data provided by various sensors such as cameras, LiDAR, ultra sonic, etc. either individually or fused together.
- Localization: this involves making an estimate of the vehicle's location withing the environment [Cad16].
- Path Planning and Control: this includes making an estimate of the trajectory of the vehicle based on the current sensor readings [Pad16].

At any sampling instant, the input signal, e.g., an image, 2-D or 3-D point cloud, has to pass through a variety processing stages before motion can be predicted. This is followed by signal processing which aims to remove noise and extract any useful information, e.g., a low pass filter is used to retain the dc component of a signal and filter out the high frequency noise. Specific applications involve image processing to remove noise from blurred images or removing outliers from a 3-D LiDAR point cloud using a RANSAC algorithm [FB81]. The perception system, with the help of processed data from sensors, builds a virtual representation of the vehicle's environment. This representation can be as partial or full as

required by the decision making algorithms further downstream in the process. Localization can also be considered a part of the perception system; because it deals with the vehicles environment as projected by the sensor data. A localization algorithm estimates the vehicle's position and pose with respect the the map/road. This can further aid in any motion planning decisions. Since this work focuses on motion control using perception, the work done on perception will be discussed in greater detail.

2.1.1 Perception

The perception system is the fundamental component in a self-driving application. It provides essential information about, or representation of, the environment. Such information is required in order to estimate the motion at the current state and, in some cases, it can be used to predict the motion at near future states. There are a number of sensors which can be used for perception systems, including: (1) light detection and ranging (LiDAR) sensors, (2) cameras and (3) radio detection and ranging (RADAR) sensors. A common shortcoming of any perception systems is their need for clean, noiseless data. In addition, in situations where computational resources are limited, data has to be compressed in order to process it in an orderly fashion.

2.1.1.1 LiDAR

LiDAR is one of the most prevailing sensors being used in the autonomous driving industry. LiDAR works by throwing out a large number of light pulses into the environment around the vehicle, it then captures its representation using the reflected pulses. The representation can be either 2-D or 3-D, depending on the LiDAR sensor used. It is quite accurate when compared to other sensors available for environmental mapping, but at the same time,

most of the available commercial LiDAR sensors are expensive. The major problem posed by a LiDAR point cloud is its sparsity. With more erratic surroundings, a LiDAR point cloud can become extremely complex. Also, LiDAR data cannot be compressed, on account that it is already providing limited available information, e.g., x,y,z coordinates of an obstacle. Hence, storing data with LiDAR can be memory expensive. As mentioned in [Pen17], there are broadly three ways to process LiDAR data:

- Point cloud based - which uses the raw sensor data. With a finer representation, this approach is expensive in terms of memory and processing time. Voxel based methods are used as improvisations to this approach [Szé10], [PL17]
- Feature based - extracting features from the LiDAR point cloud helps in dealing with data that prominently represents the key points from the environment which can aid the decision making. [Wan11], [Yu05], [Alh02]
- Grid based - converting the LiDAR point cloud into several grids using an adaptive discretization method as in [Vo15] has been shown to be successful and more memory efficient.

In addition, for LiDAR sensors with less data points per sampling such as in a 2-D LiDAR, median filter can be used to efficiently create a point cloud ready for decision making algorithms. Median filters not only work for LiDAR point clouds, but can also be used to remove noise from images. Median filters remove noise but leave intrinsic data intact. Because of its wide reach of applications, there are many toolboxes which provide inbuilt functions to perform median filtering. Due to the availability of a 2-D LiDAR, a median filter was used to denoise the point cloud in previous iteration of this work as well.

Post processing the LiDAR data, perception becomes a two fold process, which involves segregating the points from the point cloud into relevant groups and identifying their types if

needed. The former component of the pipeline is referred to as clustering (or segmentation) while the latter is called detection.

- **Segmentation** - it means clustering the data at point level or region level from the point cloud into discrete groups with each group representing a feature, object, etc. Some published works in this field revolve around edge based, region based, feature based, model based or graph based segmentation algorithms.

  - *Edge* based techniques can be applied in scenarios where there are inherent features available, such as edges (e.g., intersections and curbs on roads, door frames indoors). Although approaches presented in work well, this represents a very small subset of the environmental features a robot or an autonomous car can deal with.
  - *Region* based approaches move a step ahead and cluster the points based on a common criterion such as Euclidean distance or surface normals. As opposed to edge based approaches, region based approaches take into account the neighborhood, i.e., some key points, making edge based approaches more general purpose.
  - *Model* based segmentation eradicates the need of finding seed points in the LiDAR point cloud as an improvement over region based methods. Also, as region based approaches are dependent on the seed points alone for their accuracy, failing to select optimum key points may result in poor segmentation. Model based algorithms utilize the mathematical models lying behind attributes like planes or surfaces, shapes like spheres or cones, etc. The segmentation algorithm works by clustering the points according to a particular parameter (e.g., points satisfying the mathematical model of spherical shapes). Some of the most used algorithms

following using this method are RANSAC (Random Sample Consensus) and HT (Hough Transform). RANSAC has been also used for removing outliers from 3-D point clouds by fitting a model. Many works, including [Sch07], [LK13] show the successful use of RANSAC algorithm. Due to its efficiency, it has also been adopted by popular libraries for point cloud processing such as [RC11]. HT is the other extensively used algorithm using model based segmentation, as reviewed in [Pen17]. Work states in [Vos04] shows detection of planar surfaces in 3-D point clouds using vector normal to the surfaces. In [KM12], the HT is used to detect cylinders in the 3-D point cloud generated by LiDAR.

– *Attribute* based segmentation tries to capture the underlying attributes in the point cloud which are not replicated wholly by the spatial information alone. The success of the attribute based approach is shown in [YD13] where difficult objects like poles have been identified from within a point cloud.

– *Graph* based methods are powerful pertaining to their node-edge structure which can easily and efficiently represent an environment. Not only the representation is apt, using graph based structures also makes it easy to apply graph theory which has proven to be powerful in many domains such as image segmentation, localization and many more. In [Te18], the authors propose segmentation of point cloud using regularized graph CNN approach that exploits spectral graph theory and treats features in points as signals on the graph. Delving on similar lines, several approaches like k-nearest neighbours [GF09] have been used for segmentation of point cloud. In addition, as mentioned in [Pen17], graph based structures have been very useful in fusing multiple sensors' data together to form a whole representation of the vehicle's environment.

Apart from the above mentioned approaches, many machine learning architectures have been used to find the features from within the 3-D point cloud. While new works in the field focus on deep learning [Qi17], [Te18], [Rus08], the older works have also shown how classical machine learning can be utilized for segmentation [GF09].

- **Detection** - Post segmentation, all the clusters need to be categorized into different objects, such as road curbs, poles, walls, pedestrians, e.t.c. In [Moo09], ground clusters using histogram over normal to the surfaces. Other work was shown in [Zha13] which could also classify segments above the ground such as vegetation, building and pedestrians. The authors in [Yan18] propose a computationally efficient approach using a bird eye view of the environment instead and applying a 3-D CNN on the point cloud to detect objects pertaining to self-driving scenarios. In [Li17] however, another fully connected 3-D CNN is used to detect vehicles. The reason behind such a huge number of works using deep learning is the availability of labelled data-sets. KITTI data-set [Gei13] is one of the most extensively used data-sets for training the neural networks for object detection in point clouds. The cons of using deep learning networks however include computational complexity and memory usage. The work in [Rie17] introduces a more memory efficient Octree Network which greatly increases the input cluster resolutions.

2.1.1.2 Computer Vision and Camera

While LiDAR throws light to generate the point cloud from the objects in the environment, a camera can be used to produce either RGB (Red Blue Green) images or depth image. The VIS (visible) cameras can capture the scene by discretizing the visible spectrum into R, G and B bands, while the depth cameras use IR sensors to capture the a 3-D map of the scene.

VIS cameras, with knowledge of focal distance, can be used to perform stereoscopy vision, also called stereo vision. Cameras which also provide the depth information are also called RGBD cameras, due to an addition channel (for RGB and Depth).

Detecting objects in 2-D images has mainly been the task of machine learning techniques. With RGB images, machine learning has been one of the most used approaches to detect and classify objects in the scene. Machine learning approaches used until early 2000s involve classical machine learning architectures such as Support Vector Machines (SVM), Random Forest Classifiers, e.t.c. Later approaches involving deep networks can inherently extract features alongside the detection. Deep learning networks used extensively in the field of computer vision are MLP (Multi Layer Perceptrons) and (CNN) Convolutional Neural Networks. In context of self-driving, RNN (Recurrent Neural Networks) have also been used. In this section, we will discuss the approaches which involve classical machine learning architectures and deep neural networks.

- **Classical Machine Learning and Feature extraction**

  SVM [CV95], Random Forest Classifier [Ho95] are some of the very powerful architectures which have been used for the purpose of identifying objects and classification in images. Although they have been known to solve some of the most challenging problems in computer vision, they are not equipped to handle images directly at the input. Instead, the images are needed to be processed to extract useful features prior to inputting to any of these architectures. Rooting for this, image processing is a separate area of research which aims at feature extraction from the images. We will discuss the features relevant to the domain of self-driving which have been developed to aid computer vision with such learning methods.

  When it comes to self-driving, a perception system needs to identify key features

pertaining, but not limited to road lanes, pedestrians, road signs, traffic signals, e.t.c. There have been developed a number of tools for operating on images with different kind of filters (or kernels) to extract features and form feature descriptors (or vectors) which may represent key parameters and their location within the images. Various tried and tested filter kernels include mean filters, median filters, Gaussian smoothing, etc. Other types of filter kernels find the edges (vertical and horizontal or tilted), and can detect surfaces. The author in [Lab06] talks about detecting lanes on the roads, while outperforming previous works which would generate inaccurate results due to factors like shadows on the roads. As seen in survey [Pen17], lane feature extraction can be done by using edge detection [Ned04; Li03] and color [TETT09; Sun06a]. These feature vectors can be passed through SVM [Zha05], Random Forest Classifiers [Cho12; Xia16], or other boosted or ensemble learning methods [Bah10] to indicate the presence (or absence) of lanes [Gop12].

Other objects of interest are road signs. Work in [MB07] uses SVM for the task. Also, it is safe to mention that most of the road signs have fixed shapes (e.g., circles, triangles) and hence it would make much sense to fit such possible shapes in the image to a model and using HT [Bal81], and matching the identified shapes with signs available in the database using feature matching techniques like SIFT and SURF [Pan13], which are also often used in the process of localization.

Traffic lights are also one of the static objects which need to be identified by the perception systems for driver assistance. [DCN09] shows a work in this domain. Again, HT is used in [OO09] for feature extraction for traffic lights, while [Liu02] uses genetic algorithm with search region limits on the image to do the same.

Pedestrians and other moving vehicles on the road fall in category of dynamic objects.

Major breakthrough in pedestrian detection is HOG (Histogram of Oriented Gradients) feature descriptor [DT05b]. In this work, gradient is calculated at pixel level to generate feature vectors for edges which could represent human bodies. SVM is used to classify images as containing humans or not. In [Sun06b] Gabor filter is used to generate Gabor feature descriptor for detecting rear view vehicles. In [Wen15], the authors develop a efficient method to select Haar-like [VJ01] features using AdaBoost [Sch13] with sample's feature and its corresponding label.

- **Deep Learning**

  Deep learning has been one of the most researched topics for long time now because of its powerful capabilities in solving challenges in computer vision. One of the greatest wins for deep networks such as Convolutional Neural Networks (CNN) is their inherent feature extraction. A CNN has the capability to take a 3-D (RGB) image as its input as opposed to its classical machine learning equivalents, such as SVM. Machine learning is used to make informed decisions based on its learning from the training data provided. So in order to train a system, we need to parse the data and then pass it through the system, and comparing its output with the training labels to generate a loss metric. Deep learning does exactly the same, except it does not always require data parsing explicitly. A CNN comprises of Convolution filters which perform image processing implicitly, without having the need to have an additional image processing algorithm. The features being extracted from an image will be hidden from the user and will highly depend on the training labels provided in the training data. The authors in [Sza05] describe the use of CNN with automatic feature extraction while comparing it to SVM which uses Haar-like features for detecting

pedestrians. Similarly in [Fuk15], CNN is used for the same task while comparing it to different networks for the task of pedestrian detection. While pedestrian detection is an example, networks such as MLP and CNN can detect and classify multiple number of objects in an image if provided with good training data.

Classification tasks have been made extremely powerful with great works in the filed. With ILSVRC (ImageNet Large Scale Visual Recognition Challenge), many researchers have come together to out do previous works in object detection and classification. LeNet [LeC98], AlexNet [Kri12], VGGNet [SZ14], GoogLeNet [Sze15], ResNet [He16] are some of the great works which won the challenge with astonishing results in classification tasks. Following these benchmarks, many researchers have published their own modified version of the networks developed previously. The availability of diverse data-sets such as [Eve10], [Lin14] and [Den09] has greatly fuelled the success of these works.

In context of self-driving vehicles, object detection and localization within the image is also needed in order to make motion planning decisions. Usual approaches used in the field are creating bounding boxes around the detected objects in the image. One of the very first networks, R-CNN was presented in [Uij13], with multiple small CNN models to explore the features in sub-areas (Region of Interest) of the image and passing them through SVMs for classification purposes. An improvement over this work is Fast R-CNN, presented in [Gir15] where a single CNN was used over the image instead of multiple CNNs over multiple Regions of Interest. Yet again, the system was improved with Faster R-CNN presented in [Ren15], where a RPN (Region Proposal Network) is used to select RoIs instead of selecting multiple RoIs from the image to avoid wasteful computation. In [Dai16] R-FCN (Region based

Fully Connected Network) is used where the ResNet-101 is used to detect feasible areas in the image where objects are most likely present. All the above networks for object detection have worked well, but were not ready for deployment in real-time. A great work which worked in real-time was YOLO (You Only Look Once) [Red16]. Its improvised version, Fast YOLO [Sha17] showed great performance at 155 FPS (Frames Per Second). Another recent development is shown in SSD (Single Shot Detector) [Liu16b], which used the data-set to decide an initial guess for selecting RoIs. This approach is better as it eliminates the need of looking for RoIs from scratch. Even better versions of YOLO, YOLOv2 [Zha17] and YOLO9000 [RF17b] have outperformed all the networks with their high FPS and accuracy.

Another class of methods used for perception is image segmentation. It refers to pixel-wise classification where each pixel is classified according to the object or type of object it constitutes. [KU19] talks about various works which have been used for image segmentation for self-driving application. Image segmentation can help the perception system lay down pixel-wise labelling of lanes, vehicles, pedestrians, trees, e.t.c. present on the road. The results can then be used to perform motion planning.

2.1.2 Localization

Localization is the process of estimating the position of a robot in an environment. In addition to perception, a robot's location is required to plan a path between a source and destination. In order to reach a destination, the autonomous system needs to know where it currently is to predict the subsequent states of motion. There are various sensors involved in autonomous driving which can help the system with estimating the location. These sensors can be either odometry sensors or perception sensors or both (fusion).

- Odemetry sensors - IMU (Inertial Measurement Unit) measures the net acceleration of the robot in real world coordinates (x,y,z). The velocity and displacement from the starting point of the robot can then be derived by single and double time integration, respectively. Wheel encoders can sense the velocity of the wheels and has the capability of tracking the location with respect to starting point. In such techniques, where displacement of robot is measured using wheel displacement over time, the estimated location can be highly inaccurate due to drift in sensor readings. Kalman filter [WB95] is a recursive technique which can be used to give better estimates of the location as time evolves, reducing the uncertainty of estimation. It can also be used to fuse the data from two different sensors (e.g., IMU and Encoder) to get an even better estimate, where uncertainty of measurement of both sensors is included.

- Perception sensors - Camera, LiDAR, RADAR, e.t.c are sensors which can help the system figure out the location of the robot using the features such as landmarks in the environment. Particle filter [DM96] is a special type of Bayesian filter which use the observations of these sensors (e.g., an instantaneous or continuous LiDAR scan, an image or group of frames, e.t.c) and use existing maps to find out the location.

In many cases, especially in case of on-road vehicles, GPS (Global Positioning System) provides with a map of the environment, except in tough places like tunnels and underground roads. To tackle this, IMU mounted GPS systems have been introduced.

Apart from these situations, GPS is not expected to work in indoor environments (e.g., buildings). In this case, either a map can be built in advance, or on the fly as the robot moves and discovers the indoor environment. Rooted in solving this problem, SLAM (Simultaneous Localization And Mapping) has become a popular research area in recent years. Many SLAM algorithms have been built using individual or fusion of perception sensors and odometry.

A survey in [Bre17] states the types of SLAM as Filter based (application of Bayesian filters such as Kalman, Particle filters), Optimization based (uses Bundle Adjustment as a loss function to compare sensor readings and the map). Some of the notable filter based SLAM are EKF-SLAM [Bai06] and particle filter based [ED06]. Using BA (Bundle Adjustment), ORB-SLAM [MA15] and ORB-SLAM2 [MAT17] have been very successful. Other works such as [Tat17] employ deep learning to learn the mapping of a monocular image to the depth image associated with it to perform SLAM.

2.1.3 Path Planning and Control

Path planning and control is the final step in the pipeline as it is responsible for the final control action - the velocity of wheels to control the direction of motion. After the map has been constructed, its graph representation can be used to apply various algorithms for path planning. These algorithms work by establishing node-node connections in graph structures. Some of popular algorithms used for path planning are A* [Del09], RRT (Rapidly exploring Random Trees), RRT* , PRM (Probabilistic RoadMap) [KF11]. PRM can lay down the free route for a robot in the configurations space, while A*, RRT and RRT* can plan the path dynamically at each state of the robot's motion.

Following path planning, a control action needs to be generated. The control action may vary depending on the type of robot, here, we will talk about differential drive robots. In this case, the control action is generating wheel velocities such that the robot attains a particular velocity and moves in the planned direction. This final task can be considered as dead reckoning, where the control actions for next state are generated with respect to the previous state of the robot, and the knowledge of the map. For instance, in a previous iteration of this work, a PID (Proportional Integral Differential) controller [ÅH95] is used to

generate wheel velocities at each sampling instant, based on the perception of the robot.

CHAPTER

3

RELATED WORK AND MOTIVATION

The research reported on here was motivated by the work of NVIDIA and the early work done on this project. The idea for this work comes from [Boj16]. The former shows the implementation for an on-road self-driving car, the latter is a set of experiments using sensors other than LiDAR to enable autonomous navigation in indoor environments.

3.1 End-to-End Learning by NVIDIA

The work in [Boj16] leverages the pattern recognition capabilities of a CNN architecture, which is capable of extracting inherent feature through using convolution filters. The

features learned are highly dependent on the data examples provided. Hence, it is possible to regulate this aspect of a CNN, to have it use relevant data according to the application being built. The authors in this work develop DAVE-2 system inspired by DAVE (DARPA Autonomous Vehicle) developed in [LeC04]. The AI architecture used in this work also shares the essence of the work in [Pom89] which used a dense neural network for self-driving application.

The DAVE-2 system uses the CAN (Controller Area Network) of the vehicle to collect vehicle's steering angle, provided by the user, in association with the time-stamped video stream. Therefore, the data-set for training the CNN consists of frames and driving commands as its output. The authors mention that to make their system independent of the geometry of the vehicle, they are using the inverse of the turning radius $\frac{1}{r}$, where r is the turning radius, as the driving command. Therefore, the training data-set contains a number of frames, each with a corresponding inverse of turning radius. In addition, the authors use three cameras, one camera on either side of the vehicle, and a camera in the center. All cameras face forward. This camera configuration can accommodate any off-center shifts in image data and it allows the system to learn recovery in case of disturbances. These off-center shifts are also applied to the corresponding steering commands to ensure for robust training.

Data collection was conducted in a variety of environments and in a variety of lighting conditions. This introduced variance into the training data-set. Since the system is end-to-end, it was important for the network to see a variety of scenarios; in order to understand tasks in all environments, in settings ranging from a bright sunny day to a gloomy evening. Data was collected in scenarios where the drivers were following lanes, changing lanes and turning. The data pertaining to lane following was retained while the rest of the data was discarded. This was due to the application being used to enable lane following only using

the end-to-end approach. Also, the video stream was sampled at 10 FPS to avoid having too similar consecutive frames (and steering angles). Since there is a bias with respect to driving straight on the roads, a big proportion of data retained contained roads with curving lanes.

The CNN architecture proposed was adopted, although with some modifications in the work pertaining to this thesis. These alterations will be explained in the later sections.

For testing the trained CNN, a simulation was designed; one where the performance of the system could be evaluated. The performance metric, apart from the testing the losses of the CNN, was the duration of the experimental cycle when no human intervention was needed. The pipeline developed in the simulation or testing phases proved capable of handling human intervention, but does not take account of human intervention during the system's training cycle. Another important aspect of the simulation pipeline is the manner in which it deals with image transformation, i.e., how it account's for the vehicle moving away from it's ground truth, which essentially helps assessing the system in case of disturbances. Based on their assumption that human drivers need 6 seconds for adjusting a vehicle after disturbance on the road, the authors came up with an equation for the autonomy

$$Autonomy = 1 - \frac{6.I_N}{T_E} \tag{3.1}$$

where I_N is the number of interventions by human driver in order to correct the system in T_E seconds of autonomous driving mode.

Finally, the "mysterious" working of the CNN was also observed. As can be seen in [Boj16], the internal states of the CNN clearly extract features like lanes and lane curvature, without explicitly extracting such features at the beginning. The authors also mention that it is unclear which part of the CNN performs control and which part performs the feature

extraction.

3.2 Precursor to this Work

The work in this thesis builds on previous works into autonomous wheelchair research at NC State. A similar task of autonomous navigation was achieved using the usual pipeline involving perception, object classification, object detection and planning for motion control of an autonomous robot. The test bench used for experimentation is a Jazzy Select Wheelchair [Pri19]. .

3.2.1 Overview of the Test-Bench

The wheelchair is powered using DC motors, which are run by 2 series connected $12V$ batteries. In order to develop autonomous navigation system on the wheelchair, it was equipped with a variety of perception sensors (kinect camera [Mic18] and RPLIDAR-A1/A3 [Sla18]), development boards including Raspberry Pi and Arduino Uno, and a motor driver . Since the power supply for most of the development boards is less than $24V$, power converters were used to regulate the power supply as needed.

The structure of the wheelchair was modified to house the LiDAR and Kinect camera at locations which could be useful during perception. The LiDAR used was a 2-D scanner, which proved sufficient for operating in static surroundings; like building corridors and doorways, while the Kinect camera proved helpful for detecting objects in front of the wheelchair, both dynamic and static, like dustbins, pedestrians, e.t.c. LiDAR and kinect are connected serially to a GPU enabled laptop, which also runs the middleware ROS (Robot Operating System) [Qui09] master. Computationally intensive algorithms run on a PC laptop, while light weight computations are delegated to other development boards such

as the Raspberry Pi. ROS envelopes the entire system with different modules (and nodes, written in both Python and C++). Running the system on ROS is advantageous as it enables to synchronize data inflow and control signal outflow.

3.2.2 AUTONOMOUS NAVIGATION

Kinect and LiDAR data was interpreted separately, with their individually developed algorithms, to control the motion of the wheelchair. Pipelines for both these sensors were different, but with the common goal of sending velocity commands to the DC motors through the motor driver for controlling the motion of the wheelchair. The velocity commands were generated so as to achieve an instantaneous turning angle, based on the perceived environment.

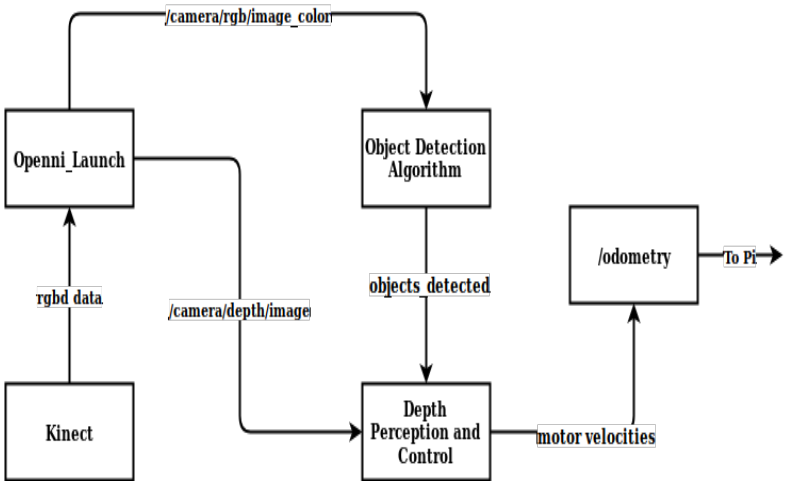

Figure 3.1 Structure of Kinect node in ROS

3.2.2.1 Object Detection and Obstacle Avoidance

The Kinect is RGBD camera. In addition to monochrome images, can also capture a depth map of the scene being viewed. For obstacle avoidance using a Kinect, both the monochrome image and the depth image are used. Readily available ROS packages like Openni_Launch enables both images to be used. Using the ROS CvBridge package, Kinect images are converted into a format which can be processed by OPENCV, an open source library available to work with images and computer vision. Also, the images can be subscribed to by different nodes within ROS on respective topics, i.e., as ROS messages. In this pipeline, two nodes working simultaneously subscribe to RGB and depth images respectively. The application of both images is described below.

- ***Object Detection*** - For object detection, RGB images were used. Here, the research published in [How17] along with the research in [Liu16a] were used. The former is a light-weight CNN architecture based on depth-wise separable convolutions. The performance of depth-wise separable convolution is shown to be $\frac{1}{N} + \frac{1}{{D_k}^2}$ times faster than regular convolution filters, where N is the number of filters in the layer and D_k is the dimension of the filter kernel. The MAP (Mean Average Precision) of MobileNet is evaluated at 88.7% on COCO data-set. Based on this performance, a decision was made to use MobileNet as the base network. In this research MobileNet runs below the SSD object detection algorithm.

  The SSD architecture builds upon the base CNN used, but it ignores the fully connected layers attached to the end of the CNN. These fully connected layers are replaced with extra convolution layers, which extract useful features at different scales and sizes. These features are then utilized for detecting multiple objects. Similar to [Sze14], the makers of SSD used the MultiBox regression technique learning to extract

the difference between ground truth bounding boxes and the predicted locations of bounding boxes. They also used this technique to estimate the confidence loss. A key feature of SSD is that it avoids the need to propose all the bounding boxes from scratch. The network learns from the data-set the most probable object locations within an image. This helps network speed and increases its efficiency. The bounding boxes that are estimated in advance are called priors.

Using a combination of SSD and MobileNet, the developed system can efficiently classify and detect objects within the scene from RGB images.

- ***Depth Perception and Control*** - There are multiple ways to estimate the distance of an object (represented by a pixel or a set of pixels) from a camera. One of the ways is using an available pin-hole camera model along with a calibration matrix. With this approach depth information can be estimated from an RGB image. The second, and more convenient way, is to use depth information directly if it is available. With the wheelchair, the Kinect camera provides the depth information directly. The depth image computed from the RGB image, which was passed through the MobileNet + SSD pipeline, was used to estimate the distance from the camera to objects (and hence from the wheelchair).

  The node working on RGB images published a custom message contains the information regarding the bounding boxes (object locations and their sizes). A separate node, which used the depth image from the Kinect camera, subscribed to this custom message and projects the locations of the bounding boxes onto the depth image. The region on the depth image which was occupied by the bounding boxes in the corresponding RGB image was considered for checking the depth value. A threshold distance of 2.5 meters was estimated, it was based on the wheelchair size and average

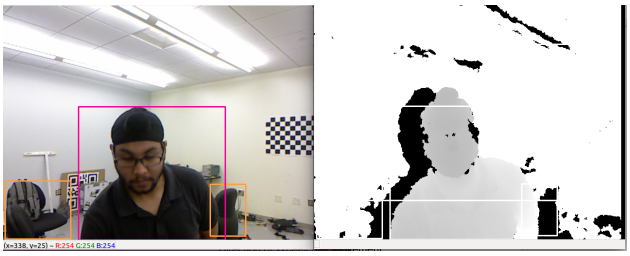

Figure 3.2 Object Detection in RGB image and corresponding Depth image

speed. Hence, the depth image was continually checking for any object(s) being closer than 2.5 meters. Object(s), if found to be closer than 2.5 meters, were avoided.

In order to control wheelchair motion during obstacle avoidance, a proportional controller only was added to the system initially. In order for the proportional controller to work an error signal was generated.

Obstacle avoidance works by removing the closest object from a specific region (called a safe zone) of the camera frame, which eventually leads to planning the path around the detected object. The safe zone is defined as a region in pixel space where no object should exist. Only then is there a safe passage of the wheelchair. Obstacles are prioritized based on the following:

- Obstacles in safe zone and closer than 2.5 meters - Highest Priority
- Obstacles in safe zone and farther than 2.5 meters - Medium Priority
- Obstacles not in safe zone - Zero Priority

Because the control algorithm is based on Proportional control only, error is defined by how much of a detected obstacle is inside the safe zone. Examples of obstacles in a depth image safe zone are shown in Fig.3.3. The center of the obstacle is calculated. If an obstacle in the left side of the safe zone, relatively, then the error is calculated as being positive (3.2a). Similarly, (3.2b) shows the calculation of error for an obstacle

in an image in the right side of the safe zone, this gives a relative negative error. The equations (3.3a) and (3.3b) represent the calculated velocity of the right and left motors of the wheelchair based on error.

$$e = obstacle_right_{edge} - safe_zone_left_{edge} \tag{3.2a}$$

$$e = obstacle_left_{edge} - safe_zone_right_{edge} \tag{3.2b}$$

$$v_r = (v_{ref} - k_p * e) * vel_{flag} \tag{3.3a}$$

$$v_l = (v_{ref} + k_p * e) * vel_{flag} \tag{3.3b}$$

where k_p is the proportional gain, e is the error calculated (in pixels) from depth image, v_{ref} is the set speed for wheelchair, v_r and v_l are the speeds calculated for right and left motors respectively, and vel_{flag} is the fail safe variable to bring the wheelchair to a complete halt. A positive error e (or obstacle on left) makes the wheelchair turn right and a negative error e (or obstacle on right) makes the wheelchair turn left.

3.2.2.2 PROXIMITY MAPPING WITH LIDAR

For this application, a 2D LiDAR was used. The top view of the LiDAR with respect to the wheelchair is shown in Fig.3.4. The direction of motion of the wheelchair aligns with a LiDAR scan data at 90°. A ROS compatible library for LiDAR called "rplidar" was used to interpret and communicate LiDAR scan data between different nodes. The package comes with a node for LiDAR which converts the raw LiDAR scan data to ROS messages. This ROS message consists of an array of values of distances between objects and LiDAR with a

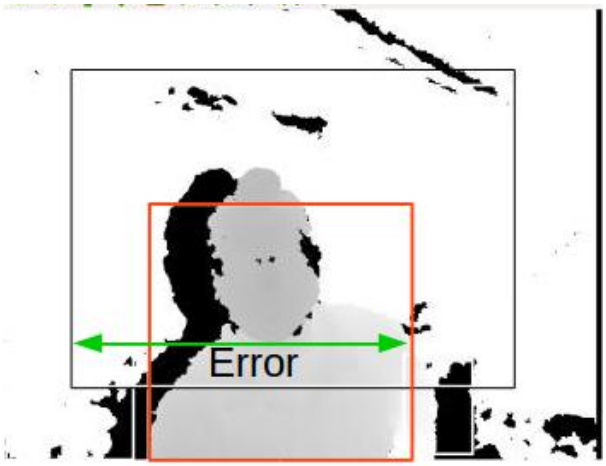

Figure 3.3 Error calculation for Depth Precision Control

resolution of 1° in a 2D plane. Based on the orientation, the field of view for this application lies between 0° and 180°. A point cloud was generated at each sampling instant, but only for objects which lie inside the wheelchair's previously defined threshold distance, i.e, 2.5m.

- ***Pre-processing of LiDAR Scan Data*** - With the help of the rplidar ROS package, a laser scan is received with values of distance between objects and LiDAR in the range 0° to 360°. The received values are not immune to noise, which calls for the need to pre-process the data before it is used for mapping the surroundings, and eventually in the control algorithm. In the pre-processing algorithm, a median filter is used to

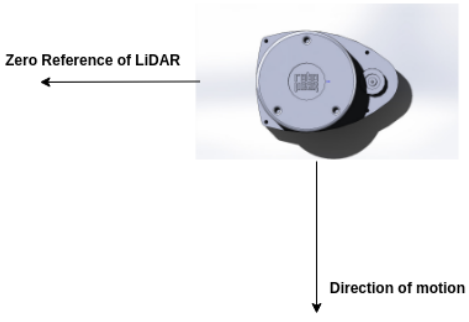

Figure 3.4 Orientation of LiDAR sensor on Wheelchair (Top View)

remove noise from the scanned data. The implementation of this filter can be seen in (3.4). After applying this filter to the raw scan data provided by the ROS node, an array of 360 accurate distance values is generated. These are the distances, d (in meters), from the wheelchair to any obstacle in its vicinity. These are the values that are used in LiDAR-based control algorithm.

$$d(k,t) = median(d(k-i, t-j)) \tag{3.4}$$

where t is the sampling instant, $k \in [0°,360°]$ is the current angle in scan at t. i, j are integers where $i \in [-2,2]$, $j \in [0,4]$. Here, scan at each angle k at sampling instant t is obtained after by considering the scan data at previous few sampling instants $(t-1, t-2, ...)$ and scan values at t at neighboring angles of k $(..., k-2, k-1, k+1, k+2, ...)$. The median of these values is taken and averaged over a few previous sampling instants to give out the filtered scan data.

- ***Motion Control with LiDAR scan point cloud*** - LiDAR is mounted at a height on the wheelchair so that it can scan moving objects, such as humans, and any static entities in enclosed spaces, such as walls in a corridor. The height was chosen such that the LiDAR is the highest sensor on the Wheelchair, so that it does not detect the patient, or the structure of the wheelchair itself. At each sampling instant, the filtered scan data is transformed into a 2D point cloud. This point cloud carries the information of the obstacles in the vicinity which are inside the threshold, at a distance of $2.5m$ or less, from the wheelchair in any direction. The point cloud was then further used to determine the clearance between every two consecutive objects, in that same manner as in [PN05]. If this calculation determines that the clearance is enough for the wheelchair to pass, a driving angle is calculated and appropriate velocity

commands are generated for the left and right wheels. In case of multiple viable directions are determined (when more than one clearance is detected), the algorithm chooses the angle which proves cheapest, i.e., the difference between the current direction and new direction is the least. If the clearance is not enough, the wheelchair is halted immediately. Variable "vel_{flag}" is used for this purpose which is set to 1 if clearance is good enough, otherwise 0. This flag provides an extra fail-safe feature to the wheelchair. A simplified blueprint of LiDAR node structure and the proximity mapping algorithm is shown in Fig.3.5. If Θ is the list of plausible directions for the wheelchair and Θ_{curr} is the current direction of wheelchair motion, then updated direction angle for wheelchair can be expressed as,

$$\Theta_{new} = \frac{\Theta_i - \Theta_{i+1}}{2}, \Theta_{new} \in \boldsymbol{\Theta} : \Delta\Theta_i = min(\boldsymbol{\Delta\Theta}) \tag{3.5}$$

where Θ_{i+1}, Θ_i are boundary angles of the clearance zone i (w.r.t. zero reference of LiDAR), $\Delta\Theta = \Theta - [\Theta_{curr}]$. Once $\Delta\Theta_i$ is calculated, the error for proportional controller can be calculated as described in (3.6).

$$e = \frac{R * \Delta\Theta_i}{W} \tag{3.6}$$

where R is the radius of wheels; W is the distance between wheel bases. The calculation of velocities can be written as (3.7a) and (3.7b).

$$v_r = (k_p * e + v_r ef) * vel_{flag} \tag{3.7a}$$

$$v_l = (k_p * e - v_r ef) * vel_{flag} \tag{3.7b}$$

where k_p, e, $v_r ef$, v_r, v_l and vel_{flag} retain their usual meanings. Since 90° of LiDAR scan aligns with direction of motion of the wheelchair and according to the zero reference chosen for LiDAR (as seen in Fig.3.4), any Θ beyond 90° represents turning left, while any Θ less than 90° represents turning right.

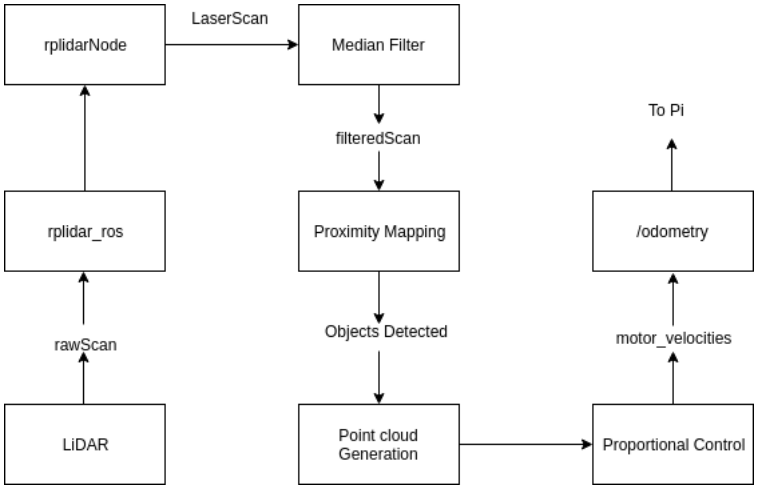

Figure 3.5 ROS node for LiDAR and Proximity Mapping algorithm

3.2.3 CHOICE OF PERCEPTION SENSOR FOR END-TO-END LEARNING

As stated above, LiDAR represents a 2-D scan of the environment, at the elevation of the LiDAR sensor in 360 points with the resolution of 1°. Also, it was mentioned in 3.1 the CNN in their system was trained against driving commands (turning radius), which is a sparse signal. When comparing the two sensors available in the research project here, 2-D LiDAR and Kinect camera, the camera is considered the better choice, because a LiDAR scan provides too sparse a perception to be learnt against the driving commands. It was found that a LiDAR scan in an indoor environment does not have as much variance when compared to both RGB and depth images captured by a Kinect camera. For these reasons,

a Kinect was chosen as the sensor for providing with input perception signals for training the CNN against the driving commands.

CHAPTER

4

SYSTEM DEVELOPMENT AND HYPOTHESIS

Based on the two pieces of work explained in chapter 3, we hypothesize that the use of the combination of MobileNet [How17] and SSD [Liu16b] for object detection and localization can be encapsulated within a single CNN for wheelchair system control (a Differential Wheel Drive Robot). Further, the approaches used appear to decrease the complexity of the perception system in terms of both design and computational expense. Hence, the system explained below was developed in order to test the hypothesis. The results of the experimental testing is shown in chapter 5.

The thrust of this work is reducing the size of the pipeline required, from perception to motion control. Optimizing the size of the pipeline was achieved by using a novel end-to-end approach. Although this work focuses on dead-reckoning using the above mentioned approach, it can also be integrated with a localization algorithm to give motion planning. Dead-reckoning is the method where an instantaneous motion is computed using the previous state of motion. In this work, the previous state of the motion was used as the reference and the instantaneous motion commands were generated from it.

For this work, the system explained in chapter 3 is modified to fit the concept of end-to-end learning using a CNN. As explained in section 3.2.3, LiDAR is no longer being used in the system for perception for this work. Our perception signal becomes RGB (or depth) image only. One of the most challenging aspects of this work was to collect data which could account for varying environments and similar activities being performed in those environments, such as driving in stipulated paths and avoiding moving pedestrians, etc. Images and corresponding driving commands were recorded synchronously to avoid additional data processing. The CNN developed in the system was expected to output the driving direction with respect to the previous state of motion of the wheelchair.

The pipeline developed is explained below, and the overall wheelchair system can be see in Fig. 4.1.

Figure 4.1 Wheelchair system

4.1 Sensors: Perception

As mentioned above, the primary perception signal is an image. Due to availability of Kinect camera, and not a 3-D LiDAR, Kinect was chosen as the main perception sensor.

4.1.1 Kinect

Using the Kinect as the primary sensor offers advantages, i.e., multiple types of images become available. Such sensors are extremely useful when dealing with algorithms that extract depth information from RGB images, e.g., SLAM algorithms. For the end-to-end approach, both RGB and depth images can be used, as both provide a fine representation of the environment in front of the wheelchair.

- ***RGB Image*** - RGB images can also be useful in detecting a variety features, whether involving depth information or not, like markings on a walls, floors or different objects in the surrounding. In this work, RGB images are primarily used to implement the end-to-end pipeline.

- ***Depth Image*** - Depth images can also provide a depth map of a scene, if one is required, thus avoiding wasteful computation in the estimation of depth from RGB images. Here, depth images were experimented with, but but they were not used. Experimentation showed, to an extent, that the approach could use depth images if necessary.

4.1.2 Webcam

The geometry of the wheelchair, and the installation of the Kinect on it, posed some challenges in relation to data collection. Due to its installation, the user's head was also captured

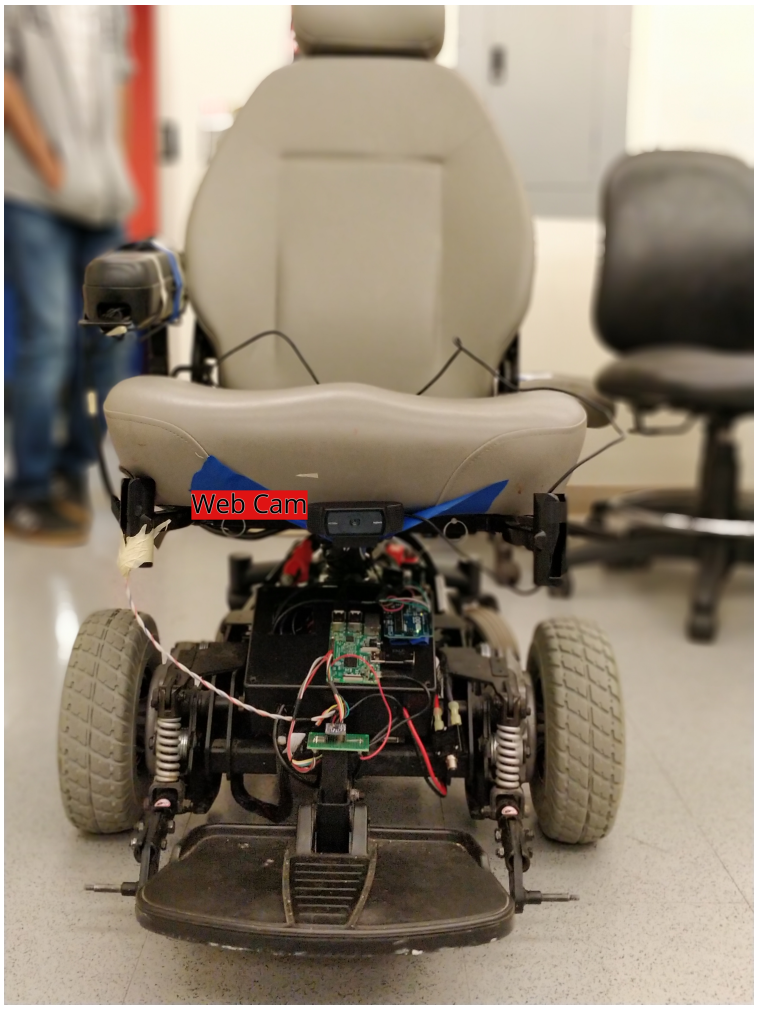

Figure 4.2 Installation of webcam on Wheelchair

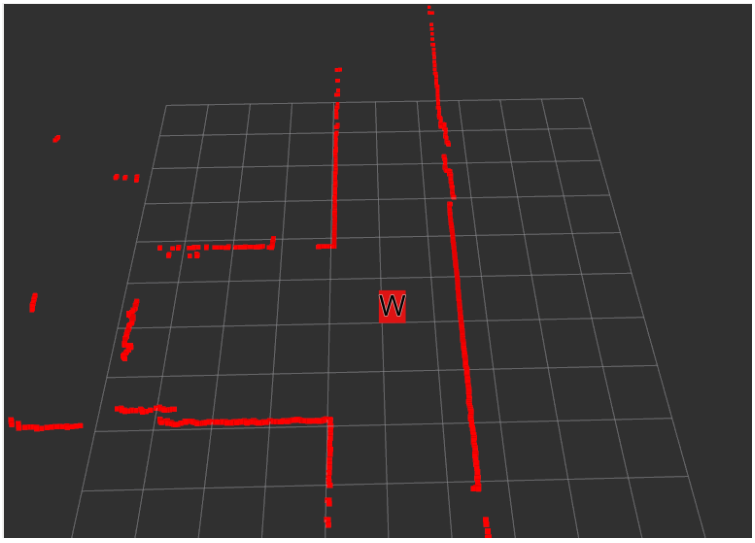

Figure 4.3 LiDAR scan of environment same as captured in Fig. 4.4

in the frame, which was absolutely unnecessary feature in the image. To solve this issue, a webcam was installed Fig.4.2 to ensure that there were no irrelevant features in the captured frame. Alternatively, the Kinect's orientation was altered, which led to objects being missed at near ground level. This is shown in Fig.4.4 where the top image shows webcam capture and the bottom shows the Kinect capture, and it is clearly seen that according to location of the webcam, objects are visible.

In addition, to defend the choice of perception sensor, it can be seen in Fig.4.3 that LiDAR scan (2-D) provides very limited data as compared to the camera. This scan was taken at the same position and orientation as in Fig.4.4

4.2 Software and Hardware Architecture

Both the software and hardware architectures are very similar to the architectures seen in previous work, see 3. The software architecture, in the ROS, was altered to accommodate

Figure 4.4 (From top) Webcam capture, Kinect capture

the range of nodes and algorithms being incorporated in this work.

4.2.1 ROS Architecture

While the previous work utilized both RGB and depth images for perception and motion control respectively, two different python nodes took care of the two algorithms separately. The system in this work utilizes only RGB image as the input to give the driving directions at the output. Therefore, a single node is used to run the CNN.

- ***Custom Messages*** - A header attribute in ROS messages carries the time-stamp associated with the message being communicated. The image messages in ROS, which in this case are being published by the Openni_kinect package already have the header. In order to train the CNN, the driving commands time had to be synchronized with the image messages. In order to do this, custom messages in ROS were made containing the driving command by the user along with the header attribute. Also, all communications between the main system and Raspberry Pi were done using a wireless TCP/IP protocol, which was available in ROS.

- ***Time Synchronizer*** - The message_filters are a utility library available in ROS. Time Synchronizer is a special kind of filter in this library, it can be used to synchronize multiple messages based on their time-stamps. The callback function in the ROS subscriber can explicitly be programmed to synchronize according to their approximate or exact same time-stamps with respective kinds of time synchronizers. In order to save images and corresponding driving commands, a time synchronizer was used so that the training of the CNN would be correct and no further data processing, i.e., at least with respect to time-stamps, was used.

- ***ROS Nodes*** - There were two modes of work involved; Training and Simulation / Testing. ROS nodes have been built to make the system as modular as possible. During the training of the CNN, a node with time-synchronizing capabilities was used to collect image and corresponding driving angle input by the user. A control node running on the Raspberry Pi (shown in 4.5) communicated the velocity commands to the wheelchair motor driver. In addition, a control node also runs in the main system, in order to generate the velocity commands from the human-operator interface, i.e., the keyboard.

  While deploying, only the node with trained CNN runs to generate the predictions for driving angle. But the Raspberry Pi runs the same control node in order to communicate the velocity commands to the motor driver.

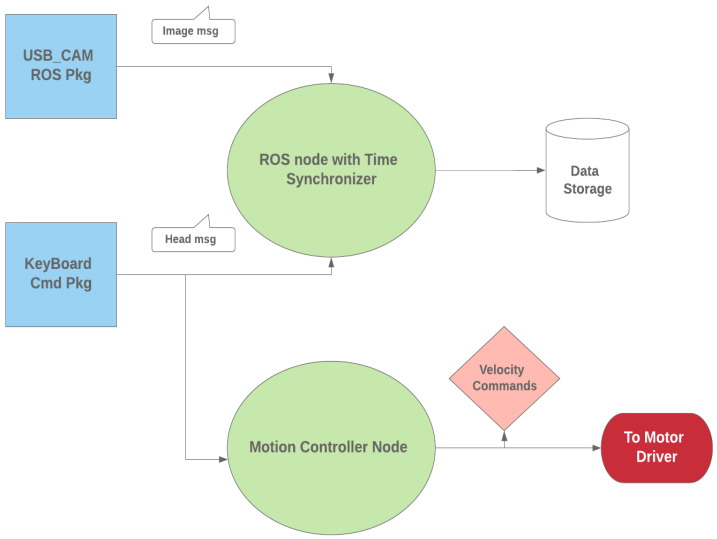

Figure 4.5 ROS architecture for training the CNN

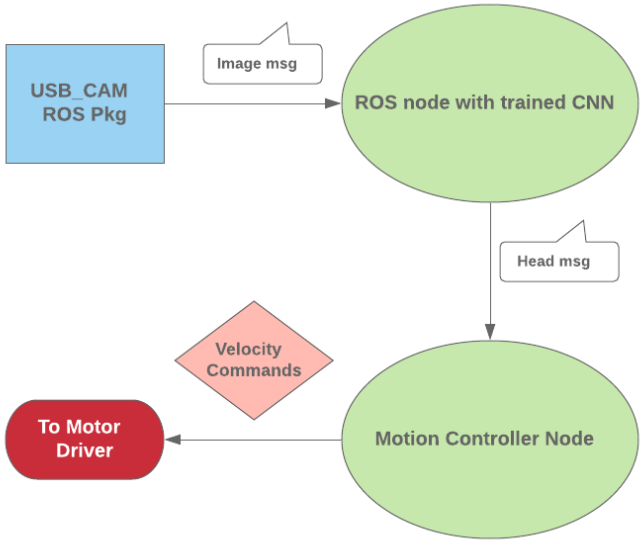

Figure 4.6 ROS architecture in deployed system

The "Head msg" signifies the driving command in figures 4.5 and 4.6.

4.2.2 Hardware pipeline

The hardware pipeline is similar to the one explained in 3.2 except for the fact that only one of the sensors is being used for perception, with 2-D LiDAR being the one which is discarded from this work due to the sparsity of its data. It can be in the Fig.4.7

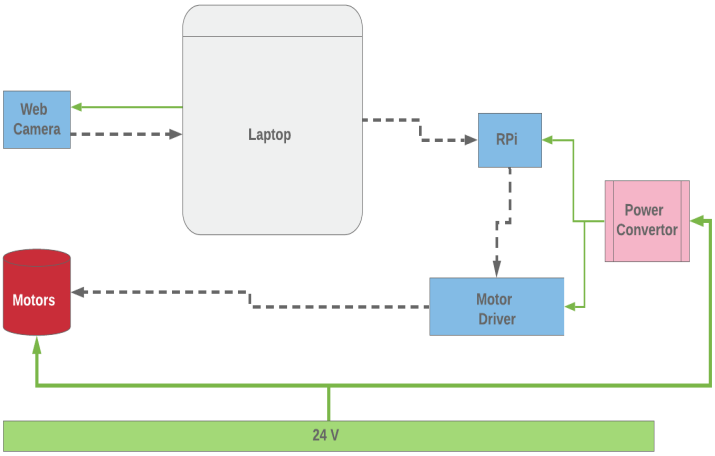

Figure 4.7 Hardware pipeline

4.3 The CNN

4.3.1 Network Architecture

The CNN architecture is adopted from [Boj16] with slight modifications. In the cited work, there is no indication of activation functions being used, but in this work, Rectified Linear Unit (ReLU) activation function has been used in the intermediate layers. In order to avoid overfitting, dropout [Sri14] layers have been included (Not shown in 4.8, 5.1).

4.3.2 Data Collection and Training

Driving command at each instant is the turning angle with respect to the direction of motion at the previous instant. It was calculated using the set of equations in (4.1)

$$\omega = \frac{v_r - v_l}{l} \tag{4.1a}$$

$$\theta = \int_t^{t+1} \omega \, d\, t \tag{4.1b}$$

where ω is the rotational velocity of the wheelchair about the ICC (Instantaneous Center of Curvature), and its time integration gives the turning angle for the wheelchair. The time integration is done by taking into account the sampling time for predictions being made by the CNN.

A C++ ROS node called the "image_listener" was built with the capability of collecting time-synchronized data and used a time synchronizer. The time synchronizer was used to collect the scene image and the corresponding driving commands.

The driving commands were fetched by tapping into the keyboard commands given by the wheelchair operator while collecting data. A python ROS node generated these commands using a package called "pygame".

The RGB images were also subscribed using an image publisher. The time-synchronizer works in a way such that only the synchronous data is collected, and the rest is discarded (hence, the message filter).

At each execution of the callback function, the synchronized data was stored to their respective locations. The driving commands (a scalar value) was saved to CSV file, while the images were stored in a directory, indexed by the order. Using this approach, training the system became fairly convenient.

The collected data was used to train the CNN. The training was not performed in one of the ROS nodes but was independent of the ROS architecture. The network was trained on over 100000 images, with activities such as following the walls in the corridors to driving

adjacent to the curb outdoors. In order to include various scenarios, a few volunteers were requested to walk in front or against the wheelchair for us to collect data while avoiding the obstacles. In addition, to account for disturbances, data was also collected starting the wheelchair in random orientations with respect to the corridor. Since there was only a single camera being used to collect the data, the only image transformation pertaining to data augmentation performed was lateral inversions. To generate corresponding driving angles Θ_a, the following equation was used

$$\Theta_a = -\Theta \tag{4.2}$$

where Θ is the driving angle before lateral inversion of the image.

4.3.3 Experimentation

The output of the CNN being developed is scalar. It is essentially the angle of turning (in degrees) with respect to the direction of motion at the previous instant. This command, can either be a floating point number or discrete value. With context to the CNN architecture, to output a floating point number a regression model has to be used with identity activation at the output layer, while for discrete value, a classification model can be built.

- ***Regression Model*** - In order to output the the floating point value for turning angle, the regression model was used. The output layer in this model does not apply any activation function. Also, this network is trained against the MSE (Means Squared Error) between the prediction and the ground truth turning angles. One interesting finding regarding this model was found inc context of training and testing metrics. The accuracy of the model was seen to be very less (around 60%), although the training and testing loss was found to be negligible. This was due to floating point values

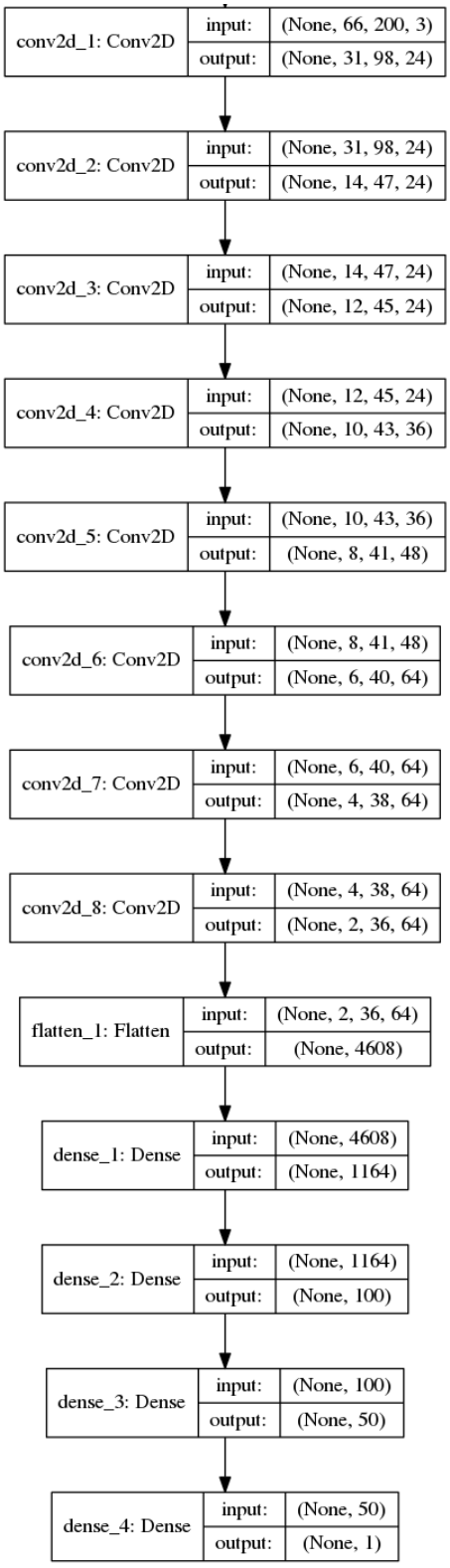

Figure 4.8 CNN architecture for Regression Model

at the output. For e.g., for the predicted turning angle of 5.35° against the ground truth turning angle of 5.4°, the metric hints at a wrong output while the error in this prediction is merely 0.9%. The regression model can be seen in Fig. 4.8.

- ***Classification Model*** - According to the geometry of the wheelchair and manual control available, the maximum turning angle input by the user was such that

$$| \theta | < 15^{\circ} \tag{4.3}$$

Keeping a margin of 5° on either direction, the region where the wheelchair can turn is in between 20° left and 20° right, with respect to the instantaneous direction of motion. For experimenting with the classification model, this region was converted to discrete zones of 1° each. Each zone was given an integer label starting at 0 to 39. The classification model would output the integer label corresponding to the zone where the wheelchair should take a turn. As compared to the regression model, the training accuracy for this model was about 89%, much higher than the former.

The input RGB image has shape of 200 x 66. Using strided convolutions, the size of the feature maps was reduced in consecutive convolution layers.

The simulations results for both the models are similar, with the regression model being better than the classification model. Simulation tests were run using both the models, and regression model was used as a primary model for deployment on the wheelchair.

4.4 Motion Control

We do not focus on generating trajectories for wheelchair. However, to demonstrate and test the performance of end-to-end learning on the real wheelchair system, a ROS node

was created so as to generate velocity commands for the instantaneous motion of the wheelchair. The ROS architecture used in the deployed system can be seen in Fig.4.6. To calculate velocity from the predicted driving angle, equations 4.4 were used.

$$v_r = v_{ref} - \Theta_p * K_p \tag{4.4a}$$

$$v_l = v_{ref} + \Theta_p * K_p \tag{4.4b}$$

where Θ_p is the predicted driving angle given by the CNN in the deployed system and v_{ref} is the the velocity of straight motion hard coded for convenience. However, it can be changed in the deployed system. K_p is the proportionality constant calibrated according to the geometry of the wheelchair.

4.5 TussleNet : Not a Net

TussleNet is not a CNN, it is just an idea used to add a feature to the pipeline explained above. In the phase of training the CNN, the pipeline was not only capable of handling human intervention during the autonomous mode of the wheelchair, it also collected data pertaining to the drastically wrong motion predictions generated our trained CNN. Authors in [Boj16] too mention handling of human intervention in autonomous mode to calculate the autonomy as shown in equation 3.1, but they do not mention training the CNN on the data where the autonomous mode went wrong.
The advantages of such a feature in the pipeline can be multiple as

- it would help the CNN get trained on some scenarios which might not have been covered in the training data-set,

- it can control overfitting of CNN by exposing it to highly variant data, and
- it is a good metric to see the real system (the wheelchair) work on predictions made by the CNN and check which kind of scenarios / environments.

CHAPTER

5

EVALUATION

As mentioned earlier in chapter 4, we aimed to shrink the size of the pipeline for perception system for the differential wheel drive robot by combining the two networks working to detect and localize objects within an image, into a single CNN which could additionally predict instantaneous driving angles.

The burden of training on the hardware and complexity of the design both go down as expected. In the previous iteration of this work as explained in chapter 3, we used MobileNet + SSD as a method for object detection ans localization, which was further used in a larger motion controlling pipeline.

5.1 Results

In [How17], the authors evaluate their proposed network with SSD on top for object detection and localization. The same combination was used in the previous iteration of this work. The number of parameters in the combined network are compared to the our network, which implicitly performs the same task, in the table 5.1. It can be seen that the number of parameters in end-to-end approach (used in this work) for the CNN are clearly lesser than the combination of MobileNet + SSD.

Network	MobileNet + SSD	End-to-End CNN
# of Params (in millions)	6.8	0.76

Table 5.1 Comparison between end-to-end and other approach

Apart from the comparison in table 5.1, there are other similar combinations which are used on top of MobileNet for object detection and localization, (shown in [How17]) such as MobileNet + Fast-RCNN (6.1 million parameters) and MobileNet + Faster-RCNN (also 6.1 million parameters). It is important to note that our work is not aimed to improve the performance of such combinations. The result above (table 5.1) signifies that these tasks can be implicitly performed by our CNN without much heavy lifting. We only present our CNN as an alternative to above mentioned networks in context of perception systems, and do not intend to out perform any of them in general.

The summary of the trained model and the number of total parameters in the network can be seen can be seen in Fig. 5.1, while the architecture of the model can be seen in Fig. 4.8. The summary is shown of the model trained in Keras [Cho15].

For visualizing the results of the trained CNN, simulation environment developed by [Lhz17] was modified to work with our system. The results can be seen in figures 5.2

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 47, 24)	14424
conv2d_3 (Conv2D)	(None, 6, 23, 24)	5208
conv2d_4 (Conv2D)	(None, 2, 11, 36)	7812
conv2d_5 (Conv2D)	(None, 1, 9, 48)	10416
conv2d_6 (Conv2D)	(None, 1, 8, 64)	6208
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 1164)	597132
dense_2 (Dense)	(None, 100)	116500
dense_3 (Dense)	(None, 50)	5050
dense_4 (Dense)	(None, 1)	51

Total params: 764,625
Trainable params: 764,625
Non-trainable params: 0

Figure 5.1 Summary for trained model

through 5.6.

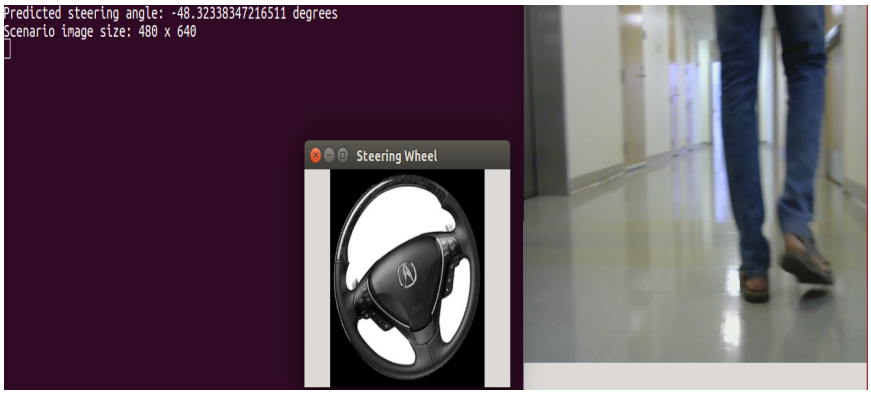

Figure 5.3 Performance scenario 2

The steering wheel indicates the number of degrees the wheelchair needs to rotate to

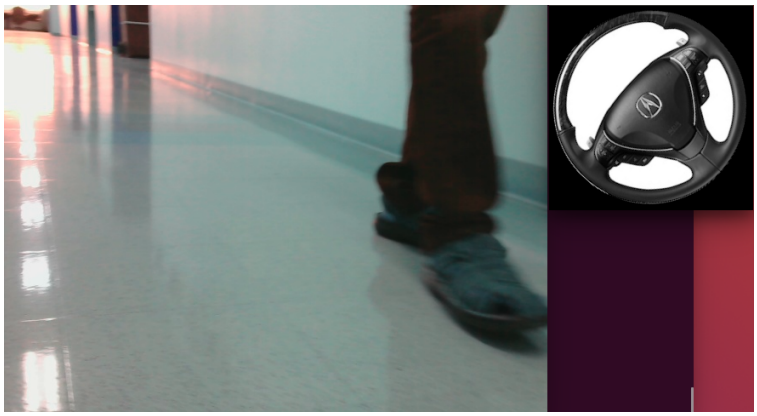

Figure 5.2 Performance scenario 1

avoid the obstacles. In figures 5.3 and 5.4, the degrees indicated in the terminal is not the pure output of the CNN. Rather, the driving angle predictions are smoothed over time, as adopted from [Lhz17].

Also, the network was trained in various scenarios including static objects and moving objects. There was no additional training provided against predicting the motion of moving objects. All the predictions only contribute to instantaneous driving commands, with no inference from previous samples or any predictions for future samples.

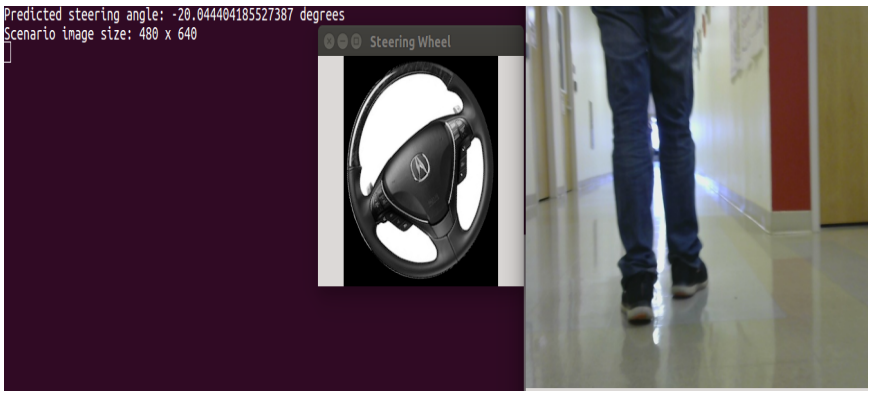

Figure 5.4 Performance scenario 3

Figure 5.5 Performance scenario 4

The system was trained mainly in indoor environments. While testing the system outdoors, it was observed that prediction of driving angles had been greatly affected. To overcome this, the data collected while testing outside was again used to train the network further. The results of simulation in outdoor environments improved as a result.

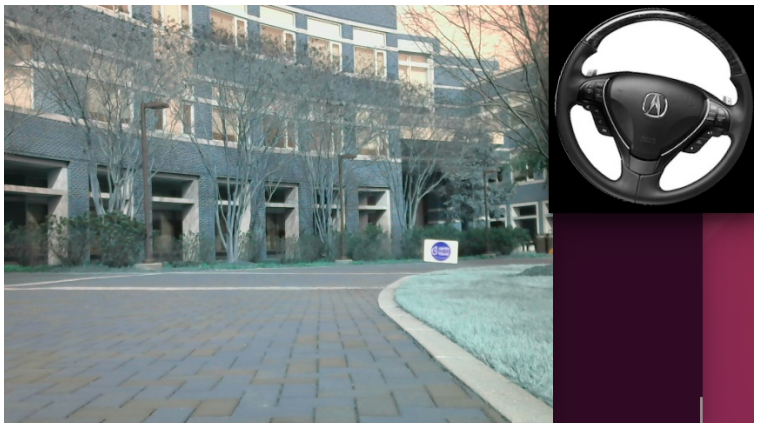

Figure 5.6 Performance scenario 5

The output of CNN is fairly accurate pertaining to the implicit features it tries to learn

from the input images. Without training the CNN for identifying any type of objects, it is observed that the network is fired up in regions where the features of importance are present. An example of this can be seen in Fig.5.7, where the top image shows the input scene image, and the image in the bottom represents the internal state of the CNN. It is evident that with proper training data-set provided, this network is trained to work as expected in an end-to-end approach. The authors in [Boj17] explain how their work in [Boj16] works by showing the internal states of the CNN, and so has been done in figures 5.8, 5.9 and 5.10 for analysis of how our network is able to decide the driving angle.

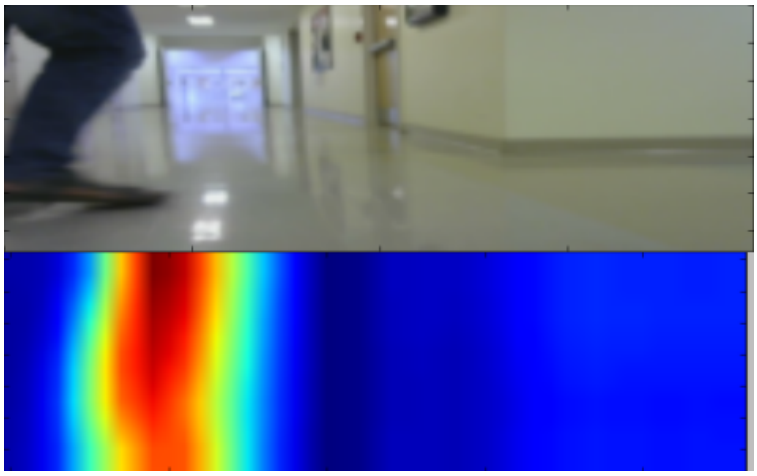

Figure 5.7 A feature map in the middle of CNN

The outputs at the end of second and fifth convolution layers can be seen in figures 5.8 and 5.10 respectively. These two images depict the indoor environment that was used to collect training data. We also observed the activation maps for the outdoor environment where we collected the training data to expose the network to varied scenarios. Fig. 5.9 shows one such example, again at the output of second convolution layer.

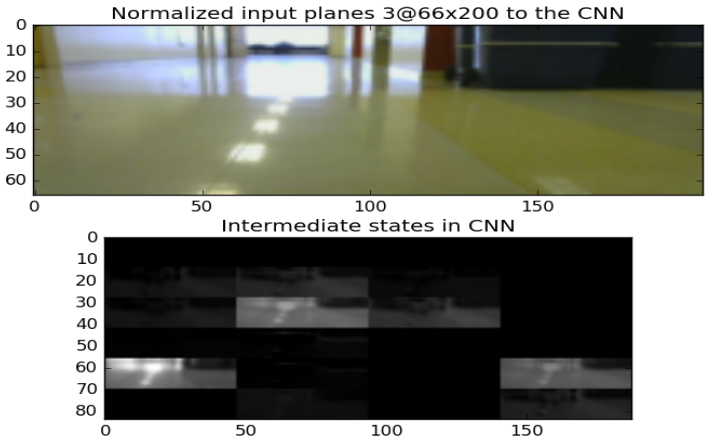

Figure 5.8 Feature maps following conv2_2d layer (indoor)

The feature maps consist of the important features which are then used in layers deeper into the network. These features, passed through successive convolution layers are reduced in dimension and finally are passed to the fully connected dense layers. As explained in [Boj16], the authors intend to perform control with the fully connected layers at the end of the CNN, while they are unsure as to which part performs feature extraction or control. Similarly, it is difficult to predict whether the true control is solely performed by fully connected layers and feature extraction by early convolution layers in our CNN. It is important to note that the CNN only predicts the driving angle, while additional control algorithms might be required to determine the trajectory of the vehicle. While we use a proportional controller, more complex and efficient algorithms can be employed when the system is deployed in tricky and highly dynamic environments for motion and path planning.

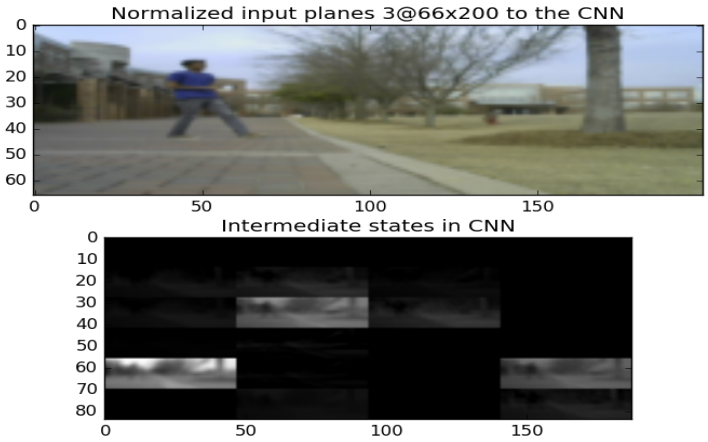

Figure 5.9 Feature maps following conv2_2d layer (outdoor)

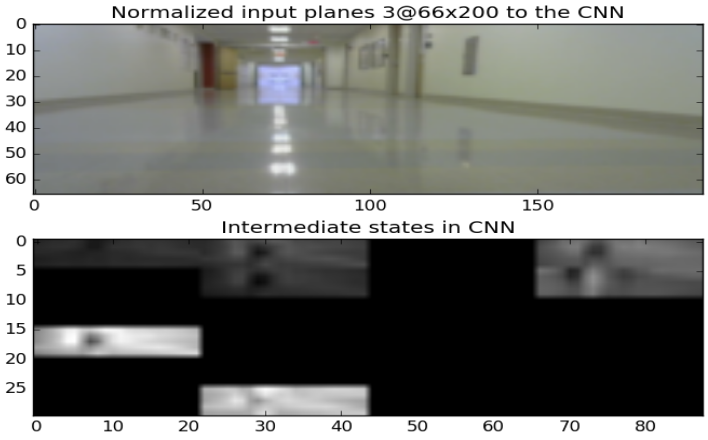

Figure 5.10 Feature maps following conv5_2d layer

It was mentioned in chapter 4 that accuracy of prediction is not a correct metric to evaluate this model as its output is a floating point number that identifies the driving angle for instantaneous motion of the wheelchair. Instead, simulation results and loss are helpful

in determining the quality of training of the network. Apart from simulation stills shown in figures 5.2 through 5.7, loss of predictions was observed while training the system, which is shown in Fig. 5.11.

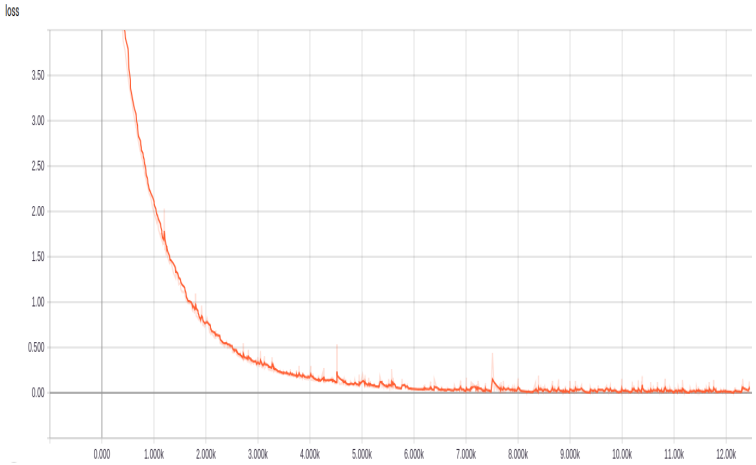

Figure 5.11 Training loss for CNN

CHAPTER

6

CONCLUSION

Perception is at the core of any autonomous driving application. Being a complex system, perception employs a number of computationally intense algorithms ranging from image processing to 3-D model reconstruction. Motivated to make our previous work (chapter 3) in similar domain more efficient, we adopted the idea put forward by [Boj16] to target the most common practices in perception.

We aimed to shrink the complexity of perception pipeline by encapsulating object detection and localization tasks into a single CNN. With a single CNN, the design of the system becomes less complex, with a significant drop in the number of parameters used by the CNN. We compared the results of this work (using the end-to-end approach) with another

approach which involves using the SSD on top of MobileNet for the same task. Without explicitly training our network to perform the task of detecting and localizing objects in an image, it predicts instantaneous driving angles for the Wheelchair used in this work, which is highly dependent on not only locations of detected objects, but also the distance between the camera and the detected object. We also showed the simulations and internal states of our CNN which explain how the network is able to output the driving angles. We also put forward the idea of TussleNet, which is an added feature to implement during the training phase, allowing us to incorporate the scenarios where our system fails to work.

With different and more varied scenarios captured by the training data, the system can be trained to work in all weather conditions and in tricky environments like city roads. With this data, the learning can be highly improved and deployed readily.

To summarize, training the CNN using end-to-end approach maps the behaviour of a human being while driving the wheelchair. This behavioural mapping takes care of objects visible in the scene, while also, their location not only in the image, but in the real environment as well. In order to avoid the obstacles, it will be required to estimate the distance between the wheelchair and the obstacle itself, which will be dependent on the wheelchair operator. We did not get a chance to collect data with many volunteers operating the wheelchair, which could rather have captured a general human behavior (for e.g., trying to avoid obstacle when distance from obstacle is less than 3 meters or 2.5 meters e.t.c.).

CHAPTER

7

FUTURE SCOPE

As mentioned in previous chapters, the work in this thesis focuses mainly on the perception system. In order to enable complete autonomy of the system, sophisticated localization and motion planning algorithms have to be incorporated into this system.

As an improvement to the perception system used in this work, section 7.1 describes another approach which can be further explored and adopted into this work. Section 7.2 talks about the addition of further useful capability more specifically relevant to our use case, the Wheelchair.

7.1 Using RNN

The CNN developed in this system with motivation from other works mentioned above is a good solution for deploying end-to-end learning for outputting driving angles. The major disadvantage of using CNNs is that they are incapable of keeping track of past samples. They can only output the instantaneous driving commands. But to think of it, driving is an activity which needs the knowledge of the state of the vehicle's environment, while also being able to generate predictions about its state for near future samples. An autonomous driving system which is capable of incorporating the states of the vehicle's scenario in the recent past, and can also predict the same for near future samples is a better solution in such an approach. For the same, RNN (Recurrent Neural Network) can be used which, instead of the instantaneous driving commands, can predict a short trajectory. This trajectory can be transformed to steering angles using motion planning algorithms. The authors in [Ban18] have developed this approach using the RNN instead of CNN.
With slight work in data processing (where each sample has outcomes in the form of a time-series) so that it is understandable by the RNN, it is quite possible for us to incorporate this approach in our work.

7.2 Integration with IoT

We presented the system embedded onto a wheelchair as the use case in this work. Apart from the perception and navigational sensors available with us for this thesis, various bio sensors have been installed on the wheelchair pertaining to the health of the patients. Furthering the work on perception system, it is planned to incorporate IoT (Internet of Things) to perform analytics on the data such as EMG (Electromyogram), EEG (Electroen-

cephalogram), ECG (Electrocardiogram), voice commands, e.t.c. collected by the sensors now available on the wheelchair. Performing analytics on such data can assist greatly in navigation as it will keep in patient's health's track over time. Furthermore, using IoT analytics, various services can be formed to detect and report anomalies in patient's health. We have been experimenting with IBM Watson IoT platform [IBM19a] , [IBM19b] to initiate our research (not in scope of this work) due to its powerful support for modern AI frameworks such as TensorFlow [Aba15]. Research is currently going on to embed cloud support to assist the real-time system for navigation, along with exploiting the IoT platform provided.

BIBLIOGRAPHY

[Aba15] Abadi, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[Alh02] Alharthy, A. "Heuristic Filtering and 3 D Feature Extraction from Lidar Data". 2002.

[ÅH95] Åström, K. J. & Hägglund, T. *PID controllers: theory, design, and tuning*. Vol. 2. Instrument society of America Research Triangle Park, NC, 1995.

[Bah10] Bahlmann, C. et al. *Method for combining boosted classifiers for efficient multi-class object detection*. US Patent 7,769,228. 2010.

[Bai06] Bailey, T. et al. "Consistency of the EKF-SLAM algorithm". *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 3562–3568.

[Baj09] Bajracharya, M. et al. "Autonomous off-road navigation with end-to-end learning for the LAGR program". *J. Field Robotics* **26** (2009), pp. 3–25.

[Bal81] Ballard, D. H. "Generalizing the Hough transform to detect arbitrary shapes". *Pattern Recognition* **13** (1981), pp. 111–122.

[Ban18] Bansal, M. et al. "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst". *arXiv preprint arXiv:1812.03079* (2018).

[Boj16] Bojarski, M. et al. "End to end learning for self-driving cars". *arXiv preprint arXiv:1604.07316* (2016).

[Boj17] Bojarski, M. et al. "Explaining how a deep neural network trained with end-to-end learning steers a car". *arXiv preprint arXiv:1704.07911* (2017).

[Bre17] Bresson, G. et al. "Simultaneous localization and mapping: A survey of current trends in autonomous driving". *IEEE Transactions on Intelligent Vehicles* **2**.3 (2017), pp. 194–220.

[Cad16] Cadena, C. et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". *IEEE Transactions on Robotics* **32** (2016), pp. 1309–1332.

[CBI18] CBInsights. *46 Corporations Working On Autonomous Vehicles*. 2018. URL: https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/.

[Cho12] Choi, J. et al. "Road identification in monocular color images using random forest and color correlogram". *International journal of automotive technology* **13**.6 (2012), pp. 941–948.

[Cho15] Chollet, F. et al. *Keras*. https://keras.io. 2015.

[CV95] Cortes, C. & Vapnik, V. "Support-Vector Networks". *Machine Learning*. 1995, pp. 273–297.

[Dai16] Dai, J. et al. "R-fcn: Object detection via region-based fully convolutional networks". *Advances in neural information processing systems*. 2016, pp. 379–387.

[DT05a] Dalal, N. & Triggs, B. "Histograms of oriented gradients for human detection". *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* **1** (2005), 886–893 vol. 1.

[DT05b] Dalal, N. & Triggs, B. "Histograms of oriented gradients for human detection". *international Conference on computer vision & Pattern Recognition (CVPR'05)*. Vol. 1. IEEE Computer Society. 2005, pp. 886–893.

[Dav15] Davies, A. *OH LOOK, MORE EVIDENCE HUMANS SHOULDN'T BE DRIVING*. 2015. URL: https://www.wired.com/2015/05/oh-look-evidence-humans-shouldnt-driving/.

[DCN09] De Charette, R. & Nashashibi, F. "Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates". *2009 IEEE Intelligent Vehicles Symposium*. IEEE. 2009, pp. 358–363.

[DM96] Del Moral, P. "Non-linear filtering: interacting particle resolution". *Markov processes and related fields* **2**.4 (1996), pp. 555–581.

[Del09] Delling, D. et al. "Engineering route planning algorithms". *Algorithmics of large and complex networks*. Springer, 2009, pp. 117–139.

[Den09] Deng, J. et al. "Imagenet: A large-scale hierarchical image database". *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[Dix18] Dixon, L. *Self Driving Car Startups to Watch*. 2018. URL: https://angel.co/job-collections/top-self-driving-cars-startups.

[ED06] Eade, E. & Drummond, T. "Scalable monocular SLAM". *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 1*. IEEE Computer Society. 2006, pp. 469–476.

[Eve10] Everingham, M. et al. "The pascal visual object classes (voc) challenge". *International journal of computer vision* **88**.2 (2010), pp. 303–338.

[FB81] Fischler, M. A. & Bolles, R. C. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". *Commun. ACM* **24** (1981), pp. 381–395.

[Fri17] Fridman, A. et al. "MIT Autonomous Vehicle Technology Study: Large-Scale Deep Learning Based Analysis of Driver Behavior and Interaction with Automation". *CoRR* **abs/1711.06976** (2017).

[Fuk15] Fukui, H. et al. "Pedestrian detection based on deep convolutional neural network with ensemble inference network". *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 223–228.

[Gei13] Geiger, A. et al. "Vision meets robotics: The KITTI dataset". *The International Journal of Robotics Research* **32**.11 (2013), pp. 1231–1237.

[Gir15] Girshick, R. "Fast r-cnn". *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[GF09] Golovinskiy, A. & Funkhouser, T. "Min-cut based segmentation of point clouds". *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE. 2009, pp. 39–46.

[Gop12] Gopalan, R. et al. "A learning approach towards detection and tracking of lane markings". *IEEE Transactions on Intelligent Transportation Systems* **13**.3 (2012), pp. 1088–1098.

[Gri18] Griswold, A. *WaymoâĂŹs self-driving car crashed because its human driver fell asleep at the wheel*. 2018. URL: https://qz.com/1410928/waymos-self-driving-car-crashed-because-its-human-driver-fell-asleep/.

[He16] He, K. et al. "Deep residual learning for image recognition". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[Ho95] Ho, T. K. “Random decision forests”. *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282.

[How17] Howard, A. G. et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. *CoRR* **abs/1704.04861** (2017).

[IBM19a] IBM. *IBM Watson Products and Services*. 2019. URL: https://www.ibm.com/watson/products-services/.

[IBM19b] IBM. *Watson IoT*. 2019. URL: https://www.ibm.com/internet-of-things.

[KF11] Karaman, S. & Frazzoli, E. “Sampling-based algorithms for optimal motion planning”. *The international journal of robotics research* **30**.7 (2011), pp. 846–894.

[KU19] Kaymak, Ç. & Uçar, A. “A Brief Survey and an Application of Semantic Image Segmentation for Autonomous Driving”. *Handbook of Deep Learning Applications*. Springer, 2019, pp. 161–200.

[KM12] Kotthäuser, T. & Mertsching, B. “Triangulation-based plane extraction for 3D point clouds”. *International Conference on Intelligent Robotics and Applications*. Springer. 2012, pp. 217–228.

[Kri12] Krizhevsky, A. et al. “Imagenet classification with deep convolutional neural networks”. *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[Lab06] Labayrade, R. et al. “A multi-model lane detector that handles road singularities”. *2006 IEEE Intelligent Transportation Systems Conference*. IEEE. 2006, pp. 1143–1148.

[LeC04] LeCun, Y et al. *DAVE: Autonomous off-road vehicle control using end-to-end learning*. Tech. rep. Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www. cs. nyu. edu/yann/research/dave/index. html, 2004.

[LeC98] LeCun, Y. et al. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* **86**.11 (1998), pp. 2278–2324.

[Lhz17] Lhzlhz. *A Deep Learning Beginner: Nvidia's End to End Learning on Steering for Self-Driving Cars*. 2017. URL: https://github.com/lhzlhz/PilotNet.

[Li17] Li, B. “3d fully convolutional network for vehicle detection in point cloud”. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1513–1518.

[Li03] Li, Q. et al. “An adaptive approach to lane markings detection”. *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*. Vol. 1. IEEE. 2003, pp. 510–514.

[Lin14] Lin, T.-Y. et al. “Microsoft coco: Common objects in context”. *European conference on computer vision*. Springer. 2014, pp. 740–755.

[Liu02] Liu, H. et al. “Real-time recognition of road traffic sign in motion image based on genetic algorithm”. *Proceedings. International Conference on Machine Learning and Cybernetics*. Vol. 1. IEEE. 2002, pp. 83–86.

[Liu16a] Liu, W. et al. “SSD: Single Shot MultiBox Detector”. *ECCV*. 2016.

[Liu16b] Liu, W. et al. “Ssd: Single shot multibox detector”. *European conference on computer vision*. Springer. 2016, pp. 21–37.

[LK13] LUCHOWSKIa, L. & KOWALSKIb, P. “Using RANSAC for 3D point cloud segmentation”. 2013.

[MB07] Maldonado-Bascón, S. et al. “Road-sign detection and recognition based on support vector machines”. *IEEE transactions on intelligent transportation systems* **8**.2 (2007), pp. 264–278.

[Mic18] Microsoft. *Kinect for Windows*. 2018. URL: https://developer.microsoft.com/en-us/windows/kinect.

[Moo09] Moosmann, F. et al. “Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion”. *2009 IEEE Intelligent Vehicles Symposium*. IEEE. 2009, pp. 215–220.

[MAT17] Mur-Artal, R. & Tardós, J. D. “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras”. *IEEE Transactions on Robotics* **33**.5 (2017), pp. 1255–1262.

[MA15] Mur-Artal, R. et al. “ORB-SLAM: a versatile and accurate monocular SLAM system”. *IEEE transactions on robotics* **31**.5 (2015), pp. 1147–1163.

[Ned04] Nedevschi, S. et al. “3D lane detection system based on stereovision”. *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749)*. IEEE. 2004, pp. 161–166.

[OO09] Omachi, M. & Omachi, S. “Traffic light detection with color and edge information”. *2009 2nd IEEE International Conference on Computer Science and Information Technology*. IEEE. 2009, pp. 284–287.

[Pad16] Paden, B. et al. “A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles”. *IEEE Transactions on Intelligent Vehicles* **1** (2016), pp. 33–55.

[Pan13] Panchal, P. et al. “A comparison of SIFT and SURF”. *International Journal of Innovative Research in Computer and Communication Engineering* **1**.2 (2013), pp. 323–327.

[Pen17] Pendleton, S. et al. “Perception, planning, control, and coordination for autonomous vehicles”. *Machines* **5**.1 (2017), p. 6.

[PL17] Plaza-Leiva, V. et al. “Voxel-Based Neighborhood for Spatial Shape Pattern Classification of Lidar Point Clouds with Supervised Learning”. *Sensors*. 2017.

[Pom89] Pomerleau, D. A. “Alvinn: An autonomous land vehicle in a neural network”. *Advances in neural information processing systems*. 1989, pp. 305–313.

[PN05] Premebida, C. & Nunes, U. “Segmentation and geometric primitives extraction from 2d laser range data for mobile robot applications”. *Robotica* **2005** (2005), pp. 17–25.

[Pri19] Pride. *Jazzy Select*. 2019. URL: https://www.pridemobility.com/p/jazzy-select-elite.

[Qi17] Qi, C. R. et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 652–660.

[Qui09] Quigley, M. et al. “ROS: an open-source Robot Operating System”. *ICRA Workshop on Open Source Software*. 2009.

[RF17a] Redmon, J. & Farhadi, A. “YOLO9000: Better, Faster, Stronger”. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 6517–6525.

[RF17b] Redmon, J. & Farhadi, A. “YOLO9000: better, faster, stronger”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[Red16] Redmon, J. et al. “You only look once: Unified, real-time object detection”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[Ren15] Ren, S. et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. *Advances in neural information processing systems*. 2015, pp. 91–99.

[Rie17] Riegler, G. et al. “Octnet: Learning deep 3d representations at high resolutions”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3577–3586.

[RC11] Rusu, R. B. & Cousins, S. “3D is here: Point Cloud Library (PCL)”. *2011 IEEE International Conference on Robotics and Automation* (2011), pp. 1–4.

[Rus08] Rusu, R. B. et al. “Towards 3D point cloud based object maps for household environments”. *Robotics and Autonomous Systems* **56**.11 (2008), pp. 927–941.

[Sch13] Schapire, R. E. “Explaining adaboost”. *Empirical inference*. Springer, 2013, pp. 37–52.

[Sch07] Schnabel, R. et al. “Efficient RANSAC for Point-Cloud Shape Detection”. *Comput. Graph. Forum* **26** (2007), pp. 214–226.

[Sha17] Shafiee, M. J. et al. “Fast YOLO: a fast you only look once system for real-time embedded object detection in video”. *arXiv preprint arXiv:1709.05943* (2017).

[SZ14] Simonyan, K. & Zisserman, A. “Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556* (2014).

[Sla18] Slamtec. *RP LiDAR A3*. 2018. URL: http://www.slamtec.com/en/lidar/a3.

[Sri14] Srivastava, N. et al. “Dropout: a simple way to prevent neural networks from overfitting”. *The Journal of Machine Learning Research* **15**.1 (2014), pp. 1929–1958.

[Sta18] Statista. *Road Accidents in the U.S. - Statistics Facts*. 2018. URL: https://www.statista.com/topics/3708/road-accidents-in-the-us/.

[Sun06a] Sun, T.-Y. et al. “HSI color model based lane-marking detection”. *2006 IEEE Intelligent Transportation Systems Conference*. IEEE. 2006, pp. 1168–1172.

[Sun06b] Sun, Z. et al. “Monocular precrash vehicle detection: features and classifiers”. *IEEE transactions on image processing* **15**.7 (2006), pp. 2019–2034.

[Sza05] Szarvas, M. et al. “Pedestrian detection with convolutional neural networks”. *Intelligent vehicles symposium*. 2005, pp. 224–229.

[Sze14] Szegedy, C. et al. “Scalable, high-quality object detection”. *arXiv preprint arXiv:1412.1441* (2014).

[Sze15] Szegedy, C. et al. “Going deeper with convolutions”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[Szé10] Székely, B. “Do we need a voxel-based approach for LiDAR data in geomorphology?” 2010.

[TETT09] Tapia-Espinoza, R. & Torres-Torriti, M. “A comparison of gradient versus color and texture analysis for lane detection and tracking”. *2009 6th Latin American Robotics Symposium (LARS 2009)*. IEEE. 2009, pp. 1–6.

[Tat17] Tateno, K. et al. “Cnn-slam: Real-time dense monocular slam with learned depth prediction”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6243–6252.

[Te18] Te, G. et al. “RGCNN: Regularized Graph CNN for Point Cloud Segmentation”. *ACM Multimedia*. 2018.

[The18] TheEconomist. *Why UberâĂŹs self-driving car killed a pedestrian*. 2018. URL: https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian.

[Uij13] Uijlings, J. R. et al. “Selective search for object recognition”. *International journal of computer vision* **104**.2 (2013), pp. 154–171.

[VJ01] Viola, P., Jones, M., et al. “Rapid object detection using a boosted cascade of simple features”. *CVPR (1)* **1** (2001), pp. 511–518.

[Vo15] Vo, A.-V. et al. “Octree-based region growing for point cloud segmentation”. *ISPRS Journal of Photogrammetry and Remote Sensing* **104** (2015), pp. 88–100.

[Vos04] Vosselman, G. et al. “Recognising structure in laser scanning point clouds”. English. *ISPRS 2004 : proceedings of the ISPRS working group VIII/2 : laser*

scanning for forest and landscape assessment, Freiburg, October 3-6, 2004. Ed. by Thies, M. et al. University of Freiburg, 2004, pp. 33–38.

[Wan11] Wang, H. et al. "Roof Detection in Lidar Data". *2011 International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping* (2011), pp. 1–5.

[WB95] Welch, G., Bishop, G., et al. "An introduction to the Kalman filter" (1995).

[Wen15] Wen, X. et al. "Efficient feature selection and classification for vehicle detection". *IEEE Transactions on Circuits and Systems for Video Technology* **25**.3 (2015), pp. 508–517.

[Woo18] Wood, L. *U.S. Autonomous Car Market 2018-2023*. 2018. URL: https://www.businesswire.com/news/home/20180831005410/en/U.S.-Autonomous-Car-Market-2018-2023---20.8.

[Xia16] Xiao, L. et al. "Monocular road detection using structured random forest". *International Journal of Advanced Robotic Systems* **13**.3 (2016), p. 101.

[Yan18] Yang, B. et al. "Pixor: Real-time 3d object detection from point clouds". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7652–7660.

[YD13] Yang, B. & Dong, Z. "A shape-based segmentation method for mobile laser scanning point clouds". *ISPRS journal of photogrammetry and remote sensing* **81** (2013), pp. 19–30.

[Yu05] Yu, T.-T. et al. "Automatic feature extraction and stereo image processing with genetic algorithms for LiDAR data". *International Conference on Computer Graphics, Imaging and Visualization (CGIV'05)* (2005), pp. 307–309.

[Zha05] Zhang, H. et al. "A novel lane detection algorithm based on support vector machine". *Progress In Electromagnetics Research Symposium*. 2005, pp. 22–26.

[Zha17] Zhang, J. et al. "A real-time chinese traffic sign detection algorithm based on modified YOLOv2". *Algorithms* **10**.4 (2017), p. 127.

[Zha13] Zhang, J. et al. "SVM-based classification of segmented airborne LiDAR point clouds in urban areas". *Remote Sensing* **5**.8 (2013), pp. 3749–3775.