TESTING AND IMPLEMENTATION OF VARIOUS SELF DRIVING CAR FEATURES USING NVIDIA JETSON NANO AND JETRACER PRO AI KIT

A PROJECT REPORT

Submitted by

SANJAY M [RA1811011010005]
DAIVIK RAJ MURALI [RA1811011010021]
HARIESH SEKAR [RA1811011010022]
R BHARATH KRISHNA [RA1811011010027]

Under the guidance of

DR.R.RAJENDRAN

(Professor, Department of Automobile Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

AUTOMOBILE ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**TESTING AND IMPLEMENTATION OF VARIOUS SELF DRIVING CAR FEATURES USING NVIDIA JETSON NANO AND JETRACER PRO AI KIT**" is the bonafide work of "**SANJAY M [RA1811011010005], DAIVIK RAJ MURALI [RA1811011010021], HARIESH SEKAR [RA1811011010022] and R BHARATH KRISHNA [RA1811011010027]**" who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

DR.R.RAJENDRAN
GUIDE
Professor
Dept. of Automobile Engineering

SIGNATURE

Dr.R.RAJENDRAN
HEAD OF THE DEPARTMENT
Dept. of Automobile Engineering

Signature of the Internal Examiner

Signature of the External Examiner

ABSTRACT

Autonomous cars, being the future of smart cars, are expected to be driverless, accident resistant and efficient. The constant rise in the number of road accidents in the past years has paved way and created the need for an ideal automobile which has the capability to perceive and understand the surroundings and act according to it on its own. Achieving this goal has been the sole purpose for automakers, as they have started working in this area in order to achieve it. Autonomous vehicles or self-driving cars are automobiles which require no human interaction or human control whatsoever according to the SAE level 5 of autonomy. Although advancements have been made in this field, there is still a requirement of human input.The main goal of our project is to implement self driving car features namely lane detection and following and obstacle avoidance using Python and its Machine learning and deep learning libraries as it is has a more simple and consistent environment with advanced AI/ML framework support. Our secondary objective would be to achieve obstacle avoidance using SLAM (Simultaneous Localization and Mapping) and ROS (Robot Operating System) with the help of a prototype which is a 1:18 model of standard car. We made use of NVIDIA JETSON NANO which is a high performance board enabled with deep learning and Artificial intelligence capabilities. We have implemented our first project objective, lane following and obstacle avoidance using Machine Learning and Deep Learning through Python and its libraries. The next goal of the project was to implement obstacle avoidance using ROS (Robot Operating System), for that we have implemented SLAM using Google Cartographer after a thorough comparison between a lot of SLAM implementation methods.

ACKNOWLEDGEMENTS

The success of any project requires involvement of many people and we are fortunate enough to get the support and guidance of the concerned people. I sincerely thank our Dean, Dr. T. V. Gopal, College of Engineering and Technology, SRM Institute of Science and Technology, Chennai. We express our thanks to our Head of the Department, Automobile engineering, Dr. Rajendran R, who has been constant source of inspiration, encouragement, guidance and the facilities provided by the department which helped us in successfully completing this project. I would like to extend our gratitude to our guide Dr.Rajendran R, Professor, Department of Automobile Engineering, Faculty of Engineering and Technology, SRM Institute of Science and Technology for his valuable guidance, consistent encouragement, personal caring, timely help and providing us with an excellent atmosphere for doing research. We would also like to thank Mr Jesu Godwin for his continuous support throughout the project. I am grateful to Mr. Jerome Stanley M, Assistant Professor and Dr. Carunai Selvane C, Assistant Professor, Department of Automobile Engineering, Faculty of Engineering and Technology, SRM Institute of Science and Technology for their continuous support throughout this project. I am also grateful to the Reviewers and the Project Evaluation Committee for their valuable guidance and suggestions throughout this project. I would like to thank my parents, without their support and encouragement this project would have not been possible.

Sanjay M

Daivik Raj Murali

Hariesh Sekar

R Bharath Krishna

TABLE OF CONTENTS

ABSTRACT **iii**

ACKNOWLEDGEMENTS **iv**

LIST OF FIGURES **ix**

ABBREVIATIONS **x**

1 INTRODUCTION **1**

1.1 General 1

1.2 Objectives 2

1.2.1 Primary Objective 2

1.2.2 Secondary Objectives 2

1.3 Need for Study 2

1.4 Organisation of the Report 3

2 LITERATURE SURVEY **4**

2.1 Introduction 4

2.2 Benchmark of Jetson 4

2.3 Deep Learning Interference Benchmarks 5

2.4 Detection 5

2.5 Lane Detection 5

2.6 Object Detection 5

2.7 Obstacle Avoidance 6

2.8 Convolutional Neural Network (CNN) 6

2.9 Sensor Fusion 7

3 HARDWARE ,COMPONENTS AND SOFTWARE **8**

3.1 Single Board Computer 8

3.2 Jetracer pro, Jetracer and Jetbot(chassis) 9
3.3 Donkey car and Jetracer 10
3.4 RPLiDAR A1 and RPLiDAR A2 10
3.5 Nvidia Jetson Nano 4GB 11
3.6 Jetracer Pro AI Kit 11
3.6.1 Contains 12
3.7 Assembly 13
3.8 RPLiDAR A1M8 14
3.8.1 LiDAR specifications 14
3.9 Li-DAR Mount 14
3.10 Base Operating System 16
3.10.1 Ubuntu 18.04 - Bionic Beaver 16
3.11 Controlling Operating System 16
3.11.1 ROS - Melodic 16
3.12 Libraries 16
3.12.1 Libraries Used 17
3.13 Packages 18
3.13.1 Packages Used 19

4 METHODOLOGY **21**
4.1 ROAD FOLLOWING 21
4.1.1 Introduction 21
4.1.2 Data Collection 21
4.1.3 Basic Motion 22
4.1.4 Teleoperation 22
4.1.5 Interactive Regression 22
4.1.6 Process 23
4.1.7 Execution 23
4.2 SIMULTANEOUS LOCALIZATION and MAPPING (SLAM) 23
4.2.1 Introduction 23
4.2.2 SLAM Packages 23
4.2.3 Hector SLAM 24

4.2.4 Gmapping 24
4.2.5 Google Cartographer 24
4.2.6 Map Generation 25
4.3 COLLISION AVOIDANCE 26
4.3.1 Introduction 26
4.3.2 Data Collection 26
4.3.3 Process 26
4.3.4 Execution 26
4.4 ERRORS 27
4.4.1 Camber Angle 27
4.4.2 Camera Error 27
4.4.3 Data 28
4.4.4 ROS Crash 28
4.4.5 Connection latency 28
4.4.6 Map Generation Overlapping 28

5 CONCLUSION AND FUTURE ENHANCEMENT 30
5.1 Conclusion 30
5.2 Future Enhancement 31
5.2.1 ROS Navigation 31
5.2.2 Sensor Fusion 31
5.2.3 Obstacle Avoidance 31
5.2.4 Behavioral Cloning 31
5.2.5 Vehicle Detection 31

A Hardware 32
A.1 Lidar Mount Drawing 32
A.2 Jetracer pro and RPLIDAR A1 specifications 32

B Codes 35
B.1 NvidiaRacecar 35
B.2 Teleoperation 36
B.3 Data Collection 37

B.4 Training Model 39
B.5 Live Execution 43

LIST OF FIGURES

3.1 Nvidia Jetson Nano 8

3.2 Raspberry Pi 9

3.3 Decision Matrix 9

3.4 RPLiDAR A1 10

3.5 RPLiDAR A2 10

3.6 nvidia jetson nano 4gb 11

3.7 Components 12

3.8 Side view 13

3.9 Front view 13

3.10 Li-DAR mount cad model 15

3.11 3D printed Li-DAR mount 15

4.1 Automotive Electronics Lab SRMIST 25

4.2 Driving Simulation LAB SRMIST 25

4.3 camera error 27

4.4 Hector SLAM error 29

4.5 Google Cartographer error 29

ABBREVIATIONS

ADAS	Advanced driver-assistance systems
API	Application Programming Interface
CNN	Convolutional Neural Network
DNN	Deep Neural Network
GPS	Global Positioning System
GPU	Graphics processing unit
LiDAR	Light Detection and Ranging
LIPO4	Lithium Phosphate
RADAR	Radio detection and ranging
ROS	Robot Operating System
RViz	ROS Visualization
SBC	Single Board Computer
SDC	Self Driving Car
SLAM	Simultaneous localization and mapping
URDF	Unified Robotics Description Format

CHAPTER 1

INTRODUCTION

1.1 General

Self driving Cars are capable of commuting from any start point to the desired destination without any human intervention.[6] [9][3].Features like ADAS, self parking and adaptive cruise control are being implemented in cars for a long time with minimal error. Recently studies and research on autonomous vehicles have been increased and a well known car manufacturer TESLA has partially achieved this task but still it requires timely input to make sure that the driver is still conscious about the driving scenario and to make sure to take control on situations.As far as the industries could achieve and implement is still Level 3 autonomous Driving but Honda has been working on Level 4 autonomous driving. Self driving cars combine a lot of sensor input to achieve self driving. Here most commonly used sensors are camera, RADAR, Odometry, GPS and Sonar but Li-DAR is also being introduced in the game for the betterment.[12] [27]

Self driving cars can be beneficial in many ways such as better vehicle to vehicle interaction which paves the way for better decisions.[38] Traffic can be reduced as much as possible so there won't be any delay in travel. Minimal vehicular accidents will occur as it connects with surrounding vehicles and their moves will be communicated precisely between the vehicles which enables this. Proper acceleration and Braking would take place which leads to better fuel economy or range. This leads to reduced driving stress as we Humans drive we should be aware of the surrounding and this might be hard for old or sick drivers but if this is done by the vehicle itself, Driving is done with minimal error and no attender needed to correct or stop the vehicle as they will be trained for all scenarios and situations. Vehicle doesn't cross the speed limit or jump traffic signals and stop points so this also leads to better and safe environment for the pedestrian, animals and the passengers.

1.2 Objectives

1.2.1 Primary Objective

- To study the various methodologies and algorithms available in the industry for deploying autonomous features in a vehicle and implement it in a scaled down model (1:18)

1.2.2 Secondary Objectives

- To implement perception, localization, planning and control features in the test model
- To execute Lane detection and following
- To generate maps of the subject environments using LiDAR and SLAM
- To drive the car autonomously without any interruption caused by objects, people and animals by implementing Obstacle avoidance

1.3 Need for Study

As per Government data driver behavior or error is a factor in 94 percent of crashes, and self-driving vehicles can help reduce driver error. Higher levels of autonomy have the ability to reduce risky and dangerous driving behaviors. The greatest promise may be reducing the devastation of impaired driving, drugged driving, unbelted vehicle occupants, speeding and distraction. High level Self-driving vehicles when on road almost have an ideal driving style , meaning less gas is burned, or battery power consumed, resulting in less air pollution. Driverless cars also mean fewer cars per household. One of the major benefits of self-driving technology is making transportation more accessible for those who cannot drive. Many people who cannot drive for various reasons, including age, disability, or lack of experience won't have a problem. Automated vehicles can be a much safer mode of transportation for these people, allowing them greater independence.

1.4 Organisation of the Report

The presentation of the project report is organised in the following order

Chapter 1

Begins with the general introduction to the Self driving cars, need for the study, objective of the project and organisation of the report

Chapter 2

Reviews the available relevant literature in the field of Hardware, sensors, features explanation and implementation, Neural networks,obstacle avoidance, real time detection, Data collection and implementation and ROS for testing and implementing the features on the scaled down model

Chapter 3

Discusses on the hardware and its comparison, components and software and packages being used and their purposes with the possible problems we might deal

Chapter 4

Discusses about the features Road Following, Simultaneous Localization and Mapping Starts and Obstacle avoidance with a explanation on what these feature are and about their functions. Followed by the explanation for the Data-set collected and how it is being used and implemented. The final part of this chapter explains the Errors we faced and their solutions

Chapter 5

Summarises the conclusion and future developments

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

In terms of research previously conducted in the field of autonomous driving, the main focus was studies related to the Jetson Nano, its role in machine learning and deep neural networks, object detected and obstacle avoidance using Li-DAR, camera and Li-DAR sensor fusion and projects conducted using the Jetracer/Jetbot/DonkeyCar. From the knowledge gained after going through specific research papers and projects, it was concluded that Nvidia's Jetson Nano is most suitable for autonomous vehicular projects due to its advanced GPU capabilities, which makes it a popular choice for Machine Learning applications.[2]

2.2 Benchmark of Jetson

For the deployment of deep learning networks, the various qualified SBCs in the market are Intel Movidius, Raspberry Pi 3+ and Nvidia Jetson platform such as Nano, TX1, TX2, and Xavier, out of which the Nvidia Jetson series is most promising especially in terms of high-speed GPU performance[31]. The features which set the Jetson series apart are;

- Energy efficient (power consumption is lesser)
- Can easily fit into almost any robot or drone due to its compact design Lesser weight
- High performance per watt and low GPU cores
- Nvidia supports cuDNN library as it is not the case for any other device

2.3 Deep Learning Interference Benchmarks

Various DNNs (deep neural networks) have been attributed specifically to the Jetson Nano for object detection, localization, image processing and recognition, etc.[29]In comparison with other platforms namely the Raspberry Pi 3/3+ and Intel Movidius, the Jetson Nano outperforms and far supersedes them. [8]

2.4 Detection

The state and perception of their surroundings are what autonomous vehicles depend or rely on, to guarantee safe and satisfying driving performance. Object detection algorithms are used by this perception system to determine pedestrians, traffic signs, other vehicles and various obstacles or barriers that a vehicle can encounter. [4]

2.5 Lane Detection

Lane detection is an essential and integral part of a autonomous vehicle's capabilities. There are numerous algorithms focused on image segmentation in order to specifically enhance line structures on a road surface and to decrease the computational power needed to process the image as well at the same time in a given image of a road or a lane. The lack of distinguishable features makes lane detection algorithms tend to be confused by other objects with similar local appearance. [14][16][34]

2.6 Object Detection

The state and perception of their surroundings are what autonomous vehicles depend or rely on, to guarantee safe and satisfying driving performance. Object detection algorithms are used by this perception system to determine pedestrians, traffic signs, other vehicles and various obstacles or barriers that a vehicle can encounter. [39] To develop a general purpose algorithm that achieves two important goals, namely high quality/accuracy and high efficiency, is known as

the ideal detection of objects. The most common problems, when it comes to object detection, are the place, the variability of objects contained in the same class is one of the huge difficulties, as well as a change in perspective, the presence of partial view, and changes in illumination, which can create shadows or reflections and cause a significant loss of information. The other issues are the problem of time efficiency, memory management and storage needed to train these detectors.[2][21]

2.7 Obstacle Avoidance

One of the most crucial features of Advanced Driver Assistance Systems (ADAS) has always been obstacle/collision avoidance.[1]The application of collision avoidance can lead to harmful or even disastrous scenarios, due to the various uncertain and unexpected possibilities on the roads. But for the development of autonomous vehicles, collision avoidance is well known as a significant trend and one of the basic requirements. One way to go about this feature of collision avoidance, is with the help of the LiDAR (Light Detection and Ranging) sensor. Path planning is recognised as one of the important steps in obstacle avoidance.[33] Path planning, simply put, is the means by which autonomous vehicles navigate through an environment by planning ahead their movements. They find a sequence of valid configurations that moves them from the source to destination.[23]

2.8 Convolutional Neural Network (CNN)

A convolutional neural network is a class of deep neural networks used for image processing as well as image recognition, which is specifically designed for processing pixel data. In simple terms, a CNN is a deep learning algorithm, that takes in an input image, assigns importance such as weights and biases which are trainable, to various aspects or objects in the image and be able to distinguish between one and the other.[13][10]Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. The layers of a CNN consist of an input layer, an output layer and a hidden layer that includes multiple convolutional layers, pooling

layers, fully connected layers and normalization layers.The resultant image which comes out would have gone through feature extraction and various objects in the image are detected and classified.[5] [24] There are various architectures of CNNs available which have been the key in building algorithms which power and shall power AI as a whole in the foreseeable future. They are namely -

- GoogLeNet
- AlexNet
- LeNet
- VGGNet
- ResNet
- ZFNet

2.9 Sensor Fusion

Sensor fusion is the process of merging data from sensors to create a more accurate conceptualization and representation of the target object, In order to widen the perception capability of an autonomous vehicle and also aiding in detection of short and long range vehicles on road.[37] Autonomous vehicles use many types of sensors namely LiDAR , Camera, RADAR ,Ultrasonic sensors and encoders and many more. Different combinations of sensors are fused to bring about varied data and information about on road situations. For example LiDAR and camera data are fused together for object detection purposes [19] [20] [7] and LiDAR and RADAR data are fused for detecting and tracking moving objects while recognizing their velocities as well.

CHAPTER 3

HARDWARE ,COMPONENTS AND SOFTWARE

3.1 Single Board Computer

A single board computer (SBC) is a device with its own operating system, memory, input/output, and other features of a fully functional computer. The two mightiest and most prominent SBCs are Nvidia's Jetson Nano and Raspberry Pi. Both these devices share a lot of common features and have been used for generic purposes like text editing, robotics and sensor interface. Although, given its size factor and power, the Jetson Nano is best suited for robotics and autonomous vehicle applications. Both the Jetson Nano and Raspberry Pi have almost the same primary features, the main difference being their respective GPUs. GPUs are considered for machine learning applications due to their parallel processing ability. The Jetson Nano, with a 128-core Maxwell GPU at 921 MHz, has a much more capable one than the Raspberry Pi 4, which makes the Jetson Nano more suitable for artificial intelligence and machine learning applications. As far as affordability, the Raspberry Pi is comparatively cheaper than the Jetson Nano. [32]

Figure 3.1: Nvidia Jetson Nano

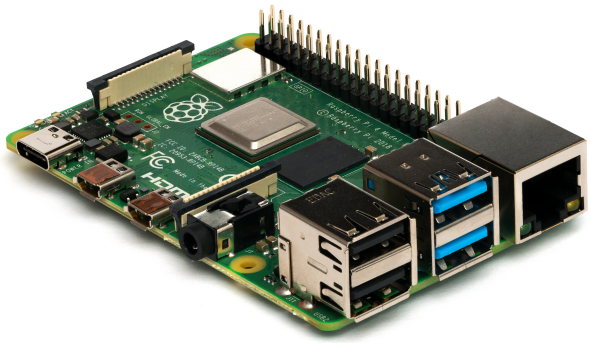


Figure 3.2: Raspberry Pi

Hardware	CPU/GPU	Memory	Online Support	Cost and Availability	Total
Raspberry pi3	2	3	5	5	15
Raspberry pi4	3	3	5	5	16
Jetson Nano	4	4	5	4	17
Jetson TX2	5	5	3	2	15

Figure 3.3: Decision Matrix

3.2 Jetracer pro, Jetracer and Jetbot(chassis)

The Jetracer is just a Jetbot on a faster RC chassis. Jetracer consists of increased speed of the software processes, which includes the camera frame rate and optimized TensorRT to handle the speed of the vehicle. It also has a simplified training scheme to collect data faster. So, just like the Jetbot, supervised learning is implemented, as in the user, put the car on the track and point to where it should drive in various different positions. This is done for about 50 times and trained for 5 min and then the car drives itself. The Jetbot uses a joystick to point to a path. But, on the JetRacer, the user need only click on the point the car should drive to. The driving type for the Jetbot is a differential two-wheel drive, whereas for the Jetracer pro, the driving type is a four-wheel drive with Ackerman steering, front and rear axle differentials. The power supply is similar for the Jetbot and Jetracer which is 12.6V, 18650 battery × 3 (connected in series) and for the Jetracer pro it is 8.4V, 18650 battery × 4 (two in parallel, two in series). Jetbot uses the program libraries of Jetbot, ROS while Jetracer uses the libraries of Jetracer, DonkeyCar. A much more experienced programming skill is required to operate the Jetracer pro when compared to Jetbot. Refer Appendix A2

3.3 Donkey car and Jetracer

A variety of hardware can be run in the Donkey car platform, which is a generalized system. The Jetracer on the other hand, is used for specific events. Unlike the DonkeyCar that uses mostly TensorFlow and command-line Python programs, the JetRacer uses the PyTorch and Jupyter Notebooks. The SBC hardware used for DonkeyCar can be Raspberry pi or Nvidia Nano even, but for Jetracer, only Nvidia Nano can be used. [15]

3.4 RPLiDAR A1 and RPLiDAR A2

The RPLiDAR performs high speed distance measurement with the difference that the A1 supports 2K/4K samples per second and A2 supports 4K/8K samples per second. Here A1 can collect data in 2 Dimension with a maximum radius of 12 meters where as A2 can collect data in 3 Dimension with a maximum radius of 18 meters. Refer Appendix A2

Figure 3.4: RPLiDAR A1

Figure 3.5: RPLiDAR A2

3.5 Nvidia Jetson Nano 4GB

The Nvidia Jetson Nano is a small, powerful computer which can run multiple neural networks in parallel for applications like image classification, object detection, speech processing, etc., all in an easy-to-use platform that runs in as little as 5 watts. With a Quad-core ARM A57 CPU, a memory of 4 GB 64-bit LPDDR4; 25.6 gigabytes/second, 128-core NVIDIA Maxwell architecture-based GPU, with LINUX as the OS support.The board is loaded with a 128GB class 10 ssd card for storage.[30]

Figure 3.6: nvidia jetson nano 4gb

3.6 Jetracer Pro AI Kit

Waveshare's Jetracer pro AI kit which is powered by Nvidia's Jetson Nano is used for Deep Learning, Self Driving and Visual Road following operations.

Figure 3.7: Components

3.6.1 Contains

The Jetracer pro developer kit comprises of the following components

- Chassis
- JetRacer Pro expansion board
- WiFi / BT AC8265 module with antennas
- IMX219-160 camera
- Acrylic mounting plate for the camera
- Camera holder
- Wireless gamepad
- Cooling fan
- 6-pin cable
- MicroSD card reader
- Wrench
- 2 x Screwdriver
- 8.4V charger + EU adapter
- Screw package
- Test route

3.7 Assembly

The Jetracer Pro kit is assembled with all the components the kit came with and the Nvidia Jetson Nano 4GB is mounted above the slave controller board(Figure3.3, Figure3.4). A cooling Fan is fixed on the heat sink. Later a LiDAR mount was fixed.

Figure 3.8: Side view

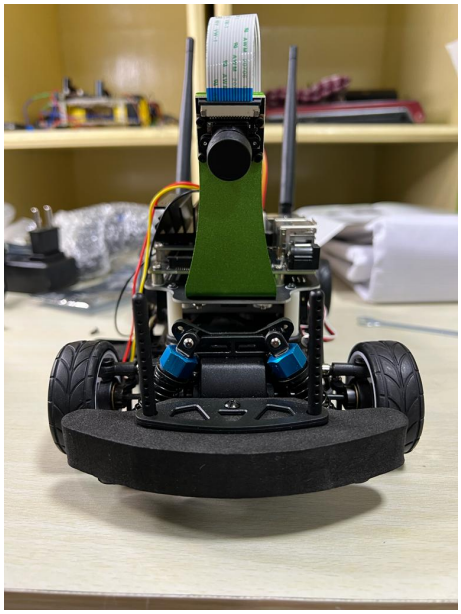


Figure 3.9: Front view

3.8 RPLiDAR A1M8

RPLiDAR A1M8: The RPLiDAR A1, which is developed by SLAMTEC, is a low cost 360-degree 2D laser scanner. This LiDAR can produce 2d point cloud data which used in mapping, localization. RPLiDAR A1 is basically a laser triangulation measurement system. It can operate in all kinds of indoor environments and outdoor environments without sunlight.

3.8.1 LiDAR specifications

The specifications are as follows;

- Detection Range: 12 meters
- 360 Degree Omnidirectional Laser Range Scanning
- 8000 Samples / Second
- Angular Resolution: 1°
- 5V Power Supply
- Supported Scan Rate: 2 – 10 Hz

3.9 Li-DAR Mount

With the help of SolidWorks a Li-DAR Mount was designed(Figure3.5) with the height of 88 mm, width of 72 mm and length of 144 mm with a allowance of 3mm for the joints. The cad model was 3d printed with PLA 3mm material and mounted on the jetracer with already existed screw holes. The RP Li-DAR is mounted on the mount using screw fixtures. Refer Appendix A1(Figure3.6)

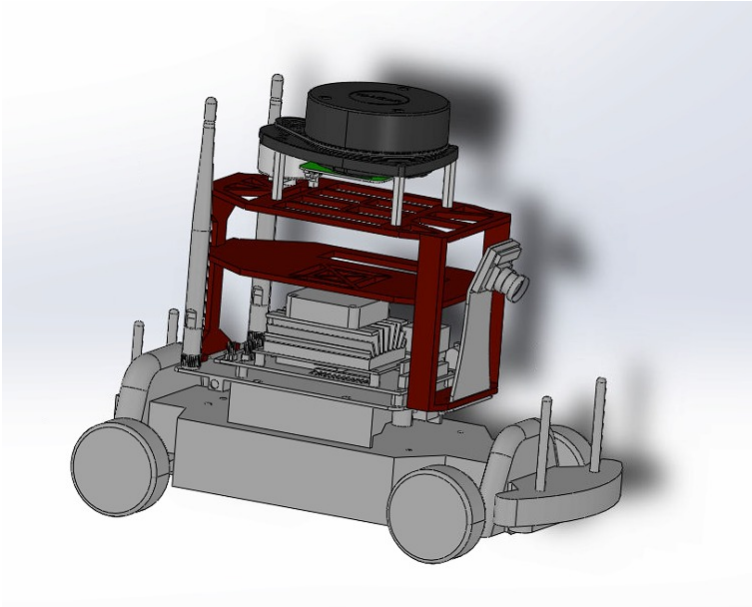


Figure 3.10: Li-DAR mount cad model

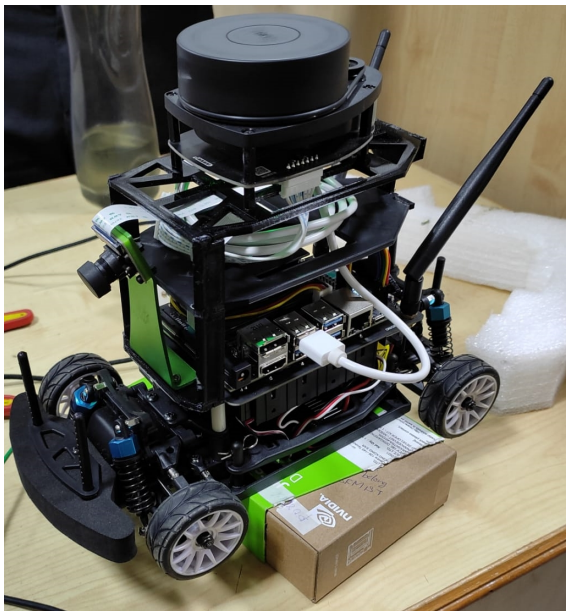


Figure 3.11: 3D printed Li-DAR mount

3.10 Base Operating System

3.10.1 Ubuntu 18.04 - Bionic Beaver

A highly usable, free and customizable alternative to the mac-OS and Windows is the Ubuntu 18.04, which is also code-named as Bionic Beaver, is a Linux based OS. This is also one of their long term support version.

3.11 Controlling Operating System

3.11.1 ROS - Melodic

Robotic Operating System (ROS) is a set of tools and software libraries which help the user build robot applications. ROS is also a global open-source community of engineers, developers and hobbyists who contribute to making robots better, more accessible and available to everyone. The current long term service version of ROS is Melodic Morenia. ROS Melodic is compatible with only Ubuntu 18.04 Bionic Beaver.

3.12 Libraries

Python libraries are built - in modules which are capable of providing many system functionalities. Few of these libraries are designed in a way to encourage and ease the portability by abstracting away platform-specifics into platform-neutral APIs.

3.12.1 Libraries Used

The Python Libraries which we used are listed below,

- Torch
- Torchvision
- Torch.nn.functional
- OpenCV (cv2)
- NumPy
- Ipywidgets
- Traitlets

Torch

It is an open source machine learning library, a script language and a scientific computing framework which uses a wide range of algorithms.

Torchvision

Torchvision is a library that goes hand in hand with PyTorch which is used for Computer Vision.

Torch.nn.functional

This library is used to apply a 3D transposed convolution operator over an input image that consists of several input planes, which is also called “deconvolution” unfold.

OpenCV (cv2)

cv2 is the module import name for OpenCV python.

NumPy

It is a python library which is implemented for data manipulation. It is used for working with arrays and also has functions for working in the domain of linear algebra, Fourier transform, and matrices.

Ipywidgets

Also known as Jupyter-widgets, ipywidgets are nothing but HTML widgets for Jupyter notebooks and the IPython kernel.

Traitlets

It is a framework which lets python classes have attributes with type checking, dynamically calculated default values, and callbacks. The library also includes a mechanism to use traitlets for configuration, loading values from files or from command line arguments.

3.13 Packages

Python and ROS are structured in small packages. These packages might contain nodes, libraries, dataset, configuration files, a third-party piece of software and other supporting materials. A particular package is limited to few designated functions as they are constructed only to execute a specific task. These packages can be created by anyone and use it by git cloning it to their repository.

3.13.1 Packages Used

The Packages which we used are listed below,

- Jetcam
- Nvidia racecar
- RPLiDAR
- Hector SLAM
- GMapping
- Google cartographer

Jetcam

Jetcam is the package which is responsible for the camera functions. From Jetcam we imported CSI camera functions because we use a CSI camera IMX219.

Nvidia racecar

Nvidia racecar is the package which functions in enabling the moving hardware with the software i.e to respond and sync to the respective code accordingly

RPLiDAR

RPLiDAR is the package which is used to initialize the Li-DAR and proceeds with point data collection and representing it in a visual form through RViz.

Hector SLAM

Hector SLAM is one of the SLAM packages which we used initially but we dropped it for the reasons discussed in chapter 6

GMapping

GMapping is one of the SLAM packages available. We tried to use it but we faced errors in the first few steps and eventually dropped this after few attempts.

Google cartographer

Google cartographer is also one of the SLAM packages available. We used this package as this was more convenient and compatible for our bot's configuration with minimum latency.

CHAPTER 4

METHODOLOGY

4.1 ROAD FOLLOWING

4.1.1 Introduction

One of the Self driving car features is road/path following. The bot is trained through data collected as images of the path chosen at various points. The goal is to train the bot in such a way that it can recognize the path used to collect the data and follow along the said path without missing the turns.[36] [17]

4.1.2 Data Collection

For creating the dataset, the images need to be manually counted and captured in the necessary conditions. The initial dataset (Dataset A) taken proved to be inefficient as the ambient brightness was much greater than what was required and the size of the data obtained (300 images) was also insufficient. While training the second dataset (Data-set B), the lighting conditions were varied and the number of images taken were increased in number resulting in a much larger data-set (500 images) compared to the first training dataset, and the test was successful.[35]

4.1.3 Basic Motion

The Jetracer is tested in this kernel in terms of steering and throttle. The Nvidia Race Car class is created here which has its two attributes - steering and throttle. The final steering value is calculated using the following equation;

y = a * x + b,

where, a is car.steering_gain

b is car.steering_offset

x is car.steering

y is the value written to the motor driver

The throttle output is computed using the equation;

y = a * x,

where, a is car.throttle_gain

x is car.throttle

y is the value written to the speed controller

4.1.4 Teleoperation

This is the kernel where the control functions are activated. The car was controlled through the game-pad which the kit came with but later for precise control of steering and throttle, Logitech g-force drive and logitech attack 3 were used respectively.

4.1.5 Interactive Regression

The camera calibration, data-set collection, model generation and the live execution for road following takes place in this kernel.

4.1.6 Process

- The CSI camera is imported from jetcam.csi_camera
- A directory is created to store the data as a task.
- During data collection, the active data-set is initialized, the widgets are created and the images are counted for and taken manually.
- With the collected data-set, the bot is trained and run. Hence, the model is created.
- The model is converted using a function, from the PyTorch format to TensorRT and run.

4.1.7 Execution

The collected data-set is processed into training and testing data (80-20% split). This split helps to validate the live input from the camera and functions the dc motor for displacement and the servo for steering as it approaches the turns and correct itself to stay within the lane boundaries.

4.2 SIMULTANEOUS LOCALIZATION and MAPPING (SLAM)

4.2.1 Introduction

Simultaneous localization and mapping(SLAM) is a function of ROS which functions to generate the Map of the subject environment and through this in autonomous vehicle localizes itself and executes the self driving functions. SLAM uses point cloud data collected with the help of a Li-DAR sensor. ROS visualise (RViz) is used to visualize the collected data in a 2 dimensional format. The generated map will be used for navigation.

4.2.2 SLAM Packages

The available SLAM packages for the installed ROS version Melodic.

- Hector SLAM
- Gmapping
- Google Cartographer

4.2.3 Hector SLAM

Hector SLAM is one of the SLAM packages available for ROS. This is the only SLAM package which doesn't rely on odometry. Hector SLAM was first introduced and used by students of TU Darmstadt. When we used Hector SLAM to generate map of our work area it seemed to work slow and as the car traveled in a bit fasters the Li-DAR couldn't hold the axis which it started and it drifted. This leads the hector slam to generate another map of the same area over the already generated map but in another orientation. Even the slowest speed of the car was too fast for Hector SLAM to create a perfect map so this SLAM method was dropped after numerous test runs.

4.2.4 Gmapping

Gmapping is another SLAM package available for ROS. This SLAM package had a issue with catkin make function. since this issue was unresolved as it has not been maintained or updated by the author this SLAM method didn't pass the initial functions

4.2.5 Google Cartographer

Google Cartographer is a SLAM package which is the only one to actively correct and generate a map with minimal errors. This package was faster than Hector SLAM but still it required to drive the car in low speeds. Cartographer doesn't print the points as it reads but took time as the car moved and turned. This was beneficial as it could correct its orientation axis and it avoided the map overlapping generation mostly. The created map when opted to save it saves twice as a pgm file and as yaml file.

4.2.6 Map Generation

We used Google cartographer to generate maps of our work area.The car was controlled through teleoperation nodes via WiFi and the controllers Logitech G-force Drive for steering and Logitech Attack 3 for acceleration. With a throttle gain of 0.30 the car was made to run and maps of the Automotive electronic lab SRMIST (Figure8.1) and Diving simulation lab SRMIST (Figure8.2) was created with minimal distortion.

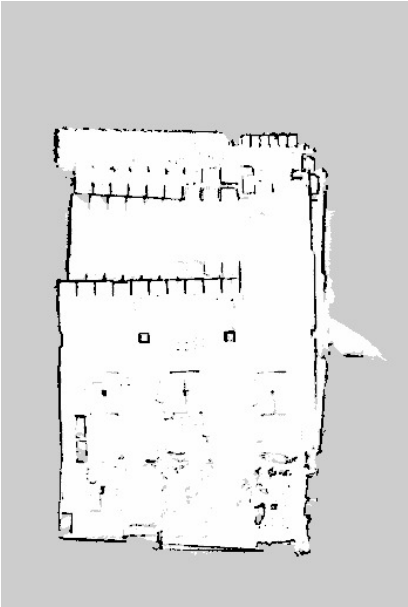


Figure 4.1: Automotive Electronics Lab SRMIST

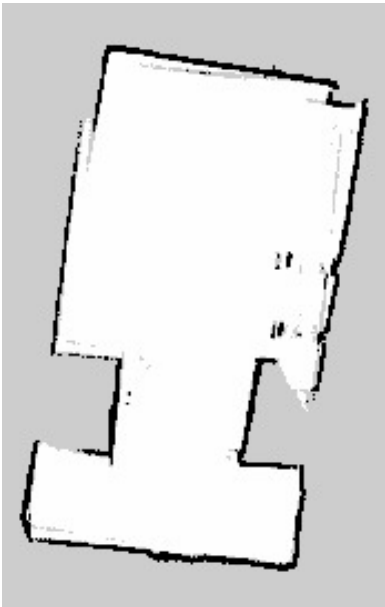


Figure 4.2: Driving Simulation LAB SRMIST

4.3 COLLISION AVOIDANCE

4.3.1 Introduction

Collision avoidance is one of the self driving car features where the car approaches an object or anything intervening the path it brakes gradually and stops itself with a certain distance between the vehicle and the object. This feature is implemented on the basis of a data-set where it has images of objects in the path.[26]

4.3.2 Data Collection

The Data were collected manually under two topics, Free way and Blocked path. Freeway is just the pictures of the lane and blocked were pictures of the lane with objects on it. Each topic had 475 pictures and hence the data-set was sufficient to execute the feature with minimum confusion for the Convolutional Neural Network.

4.3.3 Process

- Basic motion and Teleoperation are ran here for initial setup of the car.
- As mentioned earlier in the chapter 7. the CSI camera from jetcam.csi_camera is imported
- A directory is created to store the data as a task
- Along with the Road following data, the Data collected for Collision avoidance is trained and fused together.
- The model is converted using a function, from the PyTorch format to TensorRT and run.

4.3.4 Execution

The collected data along with the data collected for the road following feature are fused and trained. It again splits into training and testing data i.e 80-20 split which sums up to more than 1000 pictures on the whole. when this feature is activated the car starts to maneuver and it follows the path. since road following is also executed together when the turns are approached

the servo motor activated and turns the steering according to the subject bend(Left or Right). The maneuvering and steering are done flawlessly so in order to test collision avoidance we kept few object on the path and programmed the car in a way that when the block meter reaches to 70-80% the car would stop and when the object is removed from the path it continues to maneuver with the constant acceleration and follows the path within the lane boundaries. Hence both road following and collision avoidance were executed together successfully.

4.4 ERRORS

4.4.1 Camber Angle

Initially the jetracer had a slight offset of the rear wheels towards the right (positive camber), which was adjusted during assembly. The double wishbone on the rear-right wheel was dismantled and adjusted so that the wheel was aligned with the proper angle (camber angle).

4.4.2 Camera Error

There was a repeated error with the camera which stated that the camera could not be found, this error occurred since the camera was not terminating once we closed the kernel. So the fix for this was that to stop the camera before closing the kernel using command.

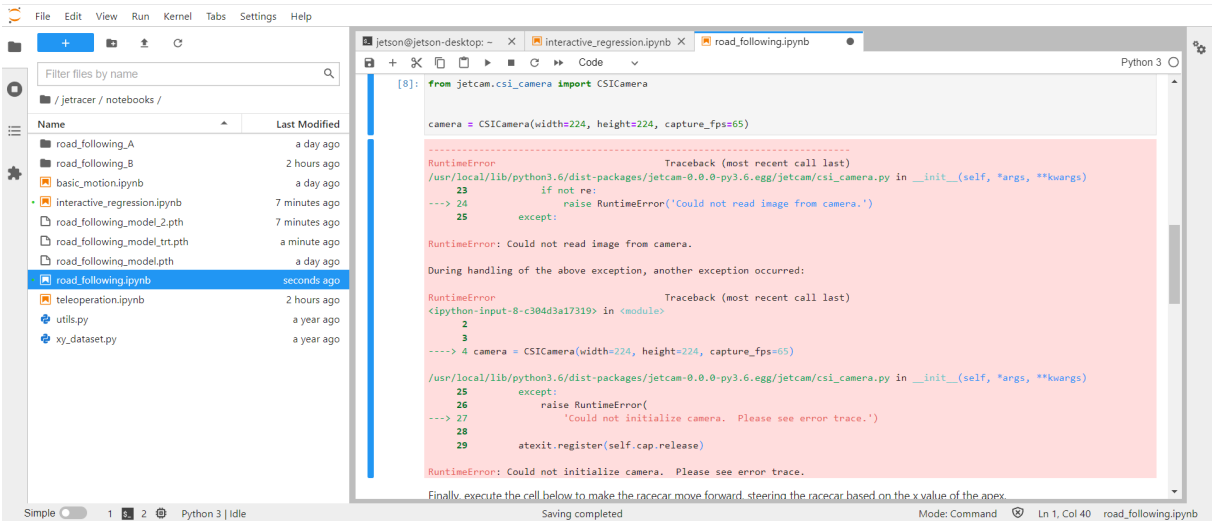


Figure 4.3: camera error

4.4.3 Data

The first data-set collected for road following proved to be ineffective. This was fixed by making a few tweaks to the surroundings, such as different lighting conditions and also, a larger amount of data was collected. The first training images taken were about 300 in number, out of which 150 were taken along the clockwise direction of the track and the rest in the opposite direction to have an even distribution of left and right turns so as to not induce bias. But the data set was insufficient and the bot took a wobbly path from the images it recognised. So a larger data-set consisting of 500-600 images was trained. Also, the lights were dimmed such that daylight alone was enough. Hence, road following was successful from the second training data set and the conditions in which it was trained.

4.4.4 ROS Crash

Multiple instances in which the ROS installation crashed. After burning the memory card a couple of times, the crashing error was fixed when the system was shutdown and restarted. After flashing the basic image in Jetson nano, we went on to installing ROS. But the installation kept failing and the reason was found to be that not all the functions were closed when we burned the memory card and repeated the image flashing and installation. So then we solved this problem by shutting down the entire system and restarting it.

4.4.5 Connection latency

There was a delay in the transfer of Li-DAR data due to the cable being too long. A shorter data cable was used to reduce the latency.

4.4.6 Map Generation Overlapping

During map generation, the speed of the jetracer was found to be crucial. Even the slightest increase in speed of the bot would throw off Hector SLAM and the map would not be recognised. As the speed of the vehicle increased, the point cloud data axis drifted to a particular angle. As an alternative, Google cartographer corrected itself when the angle drifts and was found to be

faster compared to Hector SLAM.

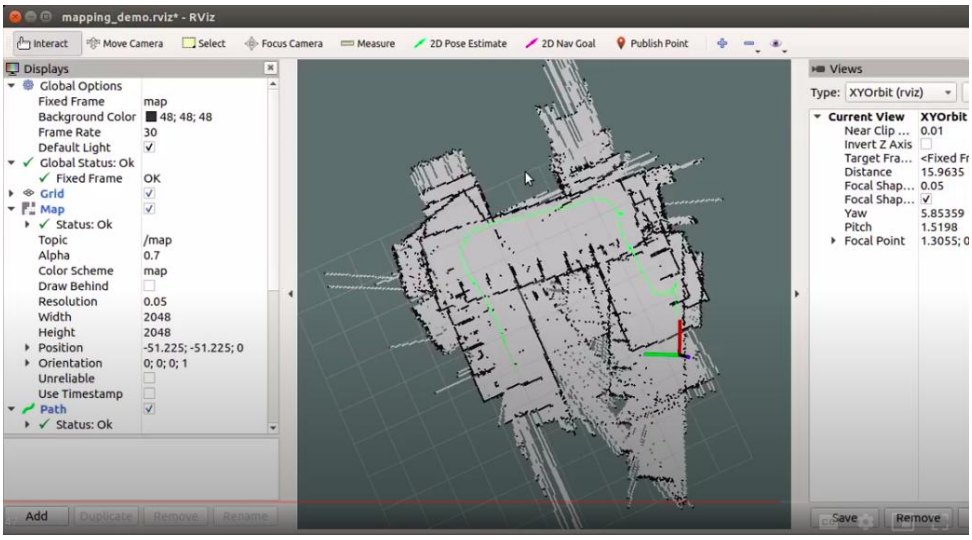


Figure 4.4: Hector SLAM error

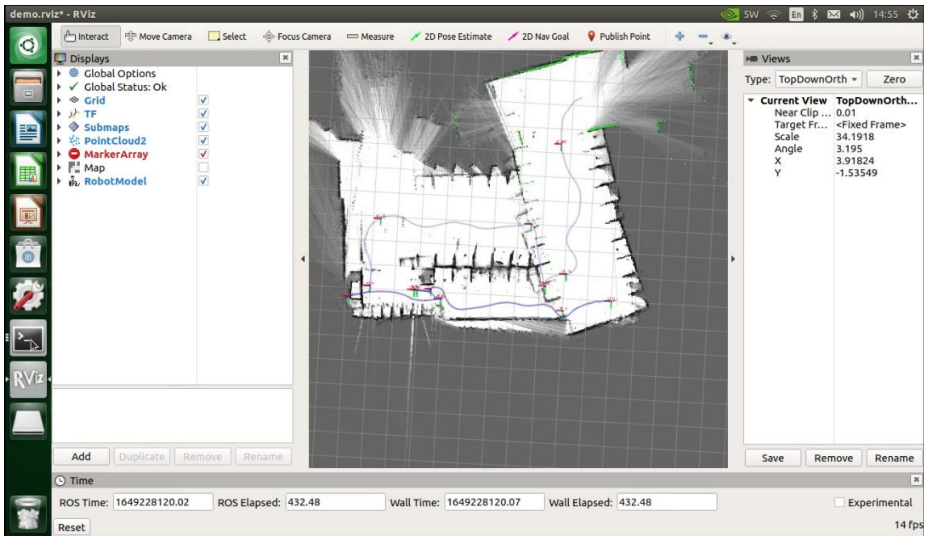


Figure 4.5: Google Cartographer error

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

After all knowledge gained from the study and research work done. ROS was installed in the bot and a thorough comparison of different available SLAM(Simultaneous localization and mapping) methods were carried out and then the optimal SLAM method was selected for use. As we had lag issues in HECTOR SLAM and Gmapping we decided to go with Google cartographer in which we could map multiple surroundings with minimal errors. The maps of our work and testing area was generated and they have been attached in chapter 4. Lane following was achieved successfully with the help of renowned python libraries namely Pytorch and OpenCV. Multiple datasets with varying number of pictures were collected in different conditions to be assured of a robust and fail-proof training model. After the dataset collection these images were fed onto the neural network and a trained model was created. This procedure was carried out along with hyperparameter tuning in each run. This trained model was then converted into a TensorRT model as TensorRT is integrated with Pytorch and Tensorflow resulting in 6x faster inference with one line of code. Finally the test run was carried out with minimal errors on the steering by collecting a data-set of 500 images in different angles,lighting and scenarios. Obstacle/collision avoidance was able to be executed after collecting multiple datasets with each data-set in the range of about 475 images for free way and blocked path each. Images where in the car has a clear path ahead was classified as free way and situations where in the path was blocked with obstacles where classified as blocked. After collecting these datasets a conditional loop was made in which the car was able to avoid collision with help of a probability distribution where after about 80% of the blocked meter indication the car stops. This equal split of data contribution laid its firm foundation for the successful execution of this Obstacle/collision avoidance feature.

5.2 Future Enhancement

Few developments, improvements and features to be achieved in the Map generation, teleoperation, behaviours and few other areas are discussed below,

5.2.1 ROS Navigation

ROS navigation is a function which enables us to control the car through ROS. Once this is done the bot can be spawned in RViz and Gazebo which will be controlled by the control nodes which would use the program structure from the ROS-Nav package.

5.2.2 Sensor Fusion

Through ROS we have planned to fuse the Li-DAR and camera data simultaneously which will pave the way for better real-time object detection, distance ranging and obstacle avoidance.[25][22][18]

5.2.3 Obstacle Avoidance

Obstacle Avoidance is the self driving car feature where the car avoids the object and drives pass through it with out any contact or pause unless any condition requires a stop.

5.2.4 Behavioral Cloning

Behavioral cloning is one of the self driving car feature which is done similar to lane detection but here as the data-set is collected the other details such as throttle input, steering angle, breaking, etc.. [28]

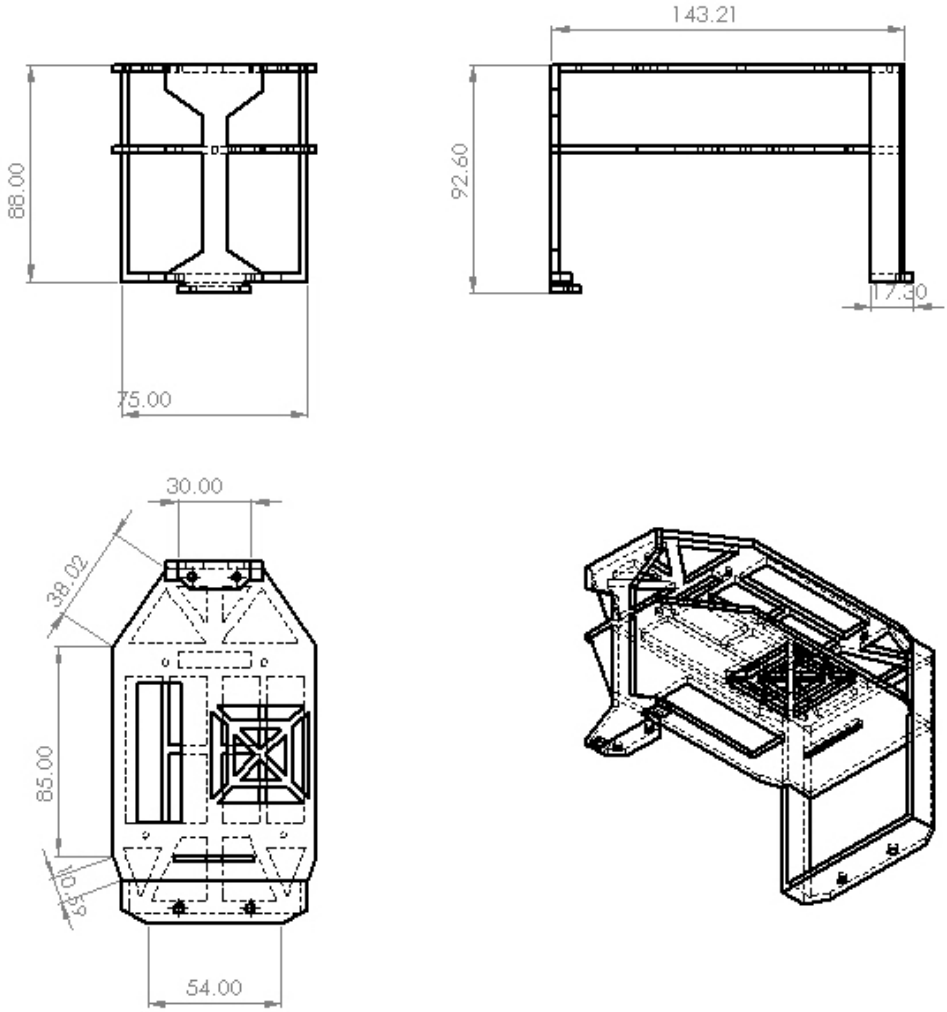
5.2.5 Vehicle Detection

Vehicle Detection is one of the self driving car feature which detect the vehicles in the surrounding adapts according to their movements for better driving conditions. [11]

APPENDIX A

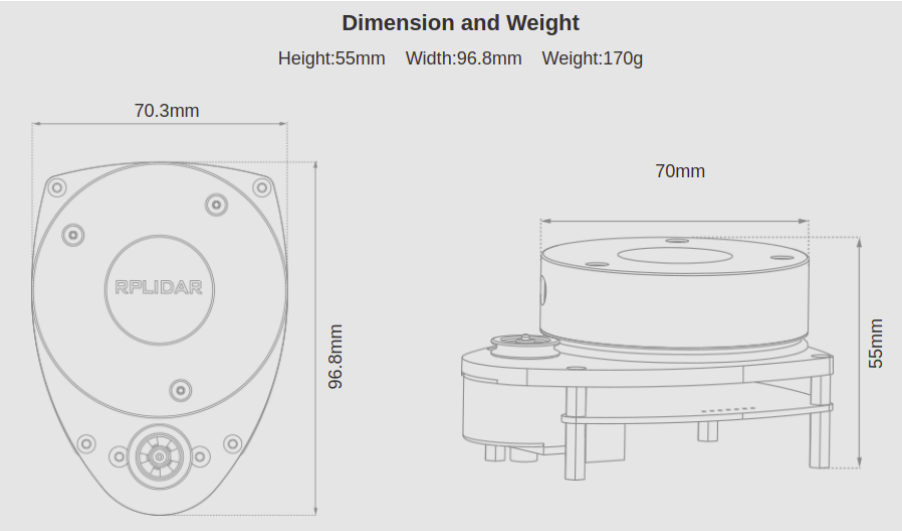
HARDWARE

A.1 Lidar Mount Drawing



A.2 Jetracer pro and RPLIDAR A1 specifications

	JETBOT		JETRACER		JETRACER PRO	
CONTROLLER*	4GB	2GB	4GB	2GB	4GB	2GB
OS	Ubuntu 18.04 LTS					
LANGUAGE	Python					
CAMERA	8MP HD resolution, 160° FOV wide angle camera					
OLED	0.91" 128×32 pixels					
WIRELESS*	AC8265	USB	AC8265	USB	AC8265	USB
TELEOPERATION	Gamepad, webpage					
COMMUNICATION	WiFi					
PROTECTION	over charge/discharge protection, over current protection, short circuit protection, reverse proof					
DRIVING TYPE	Differential 2WD		Ackerman steering Dual gearmotor rear wheel drive		Ackerman steering High speed motors 4WD Front and rear axle differentials	
SHOCK ABSORBER	N/A				Adjustable oil-filled shocks 4WD independent suspension	
POWER SUPPLY	12.6V, 18650 battery × 3 (connected in series)				8.4V, 18650 battery × 4 (two in parallel, two in series)	
	Note: the 18650 batteries are NOT included.					
MOTOR	TT motor Reduction rate 1:48 Idle speed 240RPM		37-520 DC gearmotor Reduction rate 1:10 Idle speed 740RPM		RC380 high speed carbon brushed motor Idle speed 15000RPM	
SERVO	N/A		MG996R, 9kg/cm torque		E6001, 6kg/cm torque	
CHASSIS	Aluminum alloy chassis				Quality rugged plastic chassis	
CRASHPROOF	N/A		Front sponge			
PROGRAM LIBRARY	JetBot, ROS		JetRacer, DonkeyCar			
SUGGESTION	**Entry level** Basic programming skill required		**Competition level** Experienced programming skill required		**High speed competition level** Experienced programming skill required	



Recommended applications	Vacuum robot,Home robot
Measuring Range	0.15m - 12m
Sampling Frequency	8K
Rotational Speed	5.5Hz
Angular Resolution	≤1°
Dimensions	96.8 x 70.3 x 55mm
System Voltage	5V
System Current	100mA
Power Consumption	0.5W
Output	UART Serial (3.3 voltage level)
Temperature Range	0℃-40℃
Angular Range	360°
Range Resolution	≤1% of the range（≤12m） ≤2% of the range（12m～16m）
Accuracy	1% of the range（≤3 m） 2% of the range（3-5 m） 2.5% of the range（5-25m）

APPENDIX B

CODES

B.1 NvidiaRacecar

```
from .racecar import Racecar
import traitlets
from adafruit_servokit import ServoKit
import threading

class NvidiaRacecar(Racecar):

    i2c_address = traitlets.Integer(default_value=0x40)
    steering_gain = traitlets.Float(default_value=-0.65)
    steering_offset = traitlets.Float(default_value=0.15)
    steering_channel = traitlets.Integer(default_value=0)
    throttle_gain = traitlets.Float(default_value=0.8)
    throttle_channel = traitlets.Integer(default_value=1)
    status = 0
    target = 0
    timer = None


    def __init__(self, *args, **kwargs):
        super(NvidiaRacecar, self).__init__(*args, **kwargs)
        self.kit = ServoKit(channels=16, address=self.i2c_address)
        self.kit._pca.frequency = 60
        self.steering_motor = self.kit.continuous_servo[self.steering_channel]
        self.throttle_motor = self.kit.continuous_servo[self.throttle_channel]
        self.steering_motor.throttle = 0
        self.throttle_motor.throttle = 0

    @traitlets.observe('steering')
    def _on_steering(self, change):
        self.steering_motor.throttle = change['new'] * self.steering_gain + self.steering_offset
```

```
@traitlets.observe('throttle')
def _on_throttle(self, change):
    global timer
    #self.throttle_motor.throttle = change['new'] * self.throttle_gain
    self.target = change['new'] * self.throttle_gain
    if((self.target < 0.1) and (self.target > -0.1)):
        self.status = self.target
        self.throttle_motor.throttle = self.status
    elif ((self.target < self.status) and (0 < self.status)):
        self.status = self.target
        self.throttle_motor.throttle = self.status
    else:
        self._soft_start()

def _soft_start(self):
    global timer
    if(self.status <= self.target):
        self.status = self.status + 0.18
        if(self.status >= self.target):self.status=self.target
    else:
        self.status = self.status - 0.18
        if(self.status <= self.target):self.status=self.target
    self.throttle_motor.throttle = self.status
    if(self.status != self.target):
        timer = threading.Timer(1.2, self._soft_start)
        timer.start()
```

B.2 Teleoperation

```
import ipywidgets.widgets as widgets

controller = widgets.Controller(index=0)
controller1 = widgets.Controller(index=1)# replace with index of your controller

display(controller)
display(controller1)
```

```
Controller()
```

```
Controller(index=1)
```

Even if the index is correct, you may see the text `Connect gamepad and press any button`. That's because the gamepad hasn't registered with this notebook yet. Press a button and you should see the gamepad widget appear above.

```
from jetracer.nvidia_racecar import NvidiaRacecar
import traitlets

car = NvidiaRacecar()

car.throttle_gain = 0.5
car.throttle = 0.0
```

```
car.steering_offset=0.15
car.steering = 0.01
```

```
#f_link = traitlets((controller.button[2]), (car, 'steering'), transform=lambda x: -x)
#r_link = traitlets.dlink((controller.button[1], '-0.23'), (car, 'steering'), transform=lambda x: -x)
left_link = traitlets.dlink((controller1.axes[0], 'value'), (car, 'steering'), transform=lambda x: -x)
right_link = traitlets.dlink((controller.axes[2], 'value'), (car, 'throttle'), transform=lambda x: -x)
```

B.3 Data Collection

Camera

```
from jetcam.csi_camera import CSICamera
from jetcam.utils import bgr8_to_jpeg

camera = CSICamera(width=224, height=224)
camera.running = True
```

Creating Directory for Data Collection

```
import torchvision.transforms as transforms
from xy_dataset import XYDataset
import os

TASK = 'road_following'
CATEGORIES = ['apex']
DATASETS = ['A', 'B']

TRANSFORMS = transforms.Compose([
    transforms.ColorJitter(0.2, 0.2, 0.2, 0.2),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

datasets = {}
for name in DATASETS:
    datasets[name] = XYDataset(TASK + '_' + name, CATEGORIES, TRANSFORMS, random_hflip=True)

blocked_dir = 'dataset/blocked'
free_dir = 'dataset/free'
try:
    os.makedirs(free_dir)

    os.makedirs(blocked_dir)
except FileExistsError:
    print('Directories not created because they already exist')
```

Data Collection For Road Following

```
import cv2
import ipywidgets
import traitlets
from IPython.display import display
from jetcam.utils import bgr8_to_jpeg
from jupyter_clickable_image_widget import ClickableImageWidget

dataset = datasets[DATASETS[0]]
camera.unobserve_all()

camera_widget = ClickableImageWidget(width=camera.width, height=camera.height)
snapshot_widget = ipywidgets.Image(width=camera.width, height=camera.height)
traitlets.dlink((camera, 'value'), (camera_widget, 'value'), transform=bgr8_to_jpeg)

dataset_widget = ipywidgets.Dropdown(options=DATASETS, description='dataset')
category_widget = ipywidgets.Dropdown(options=dataset.categories, description='category')
count_widget = ipywidgets.IntText(description='count')

count_widget.value = dataset.get_count(category_widget.value)

def set_dataset(change):
    global dataset
    dataset = datasets[change['new']]
    count_widget.value = dataset.get_count(category_widget.value)
dataset_widget.observe(set_dataset, names='value')

def update_counts(change):
    count_widget.value = dataset.get_count(change['new'])
category_widget.observe(update_counts, names='value')
```

```
def save_snapshot(_, content, msg):
    if content['event'] == 'click':
        data = content['eventData']
        x = data['offsetX']
        y = data['offsetY']
        dataset.save_entry(category_widget.value, camera.value, x, y)
        snapshot = camera.value.copy()
        snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
        snapshot_widget.value = bgr8_to_jpeg(snapshot)
        count_widget.value = dataset.get_count(category_widget.value)

camera_widget.on_msg(save_snapshot)

data_collection_widget = ipywidgets.VBox([
    ipywidgets.HBox([camera_widget, snapshot_widget]),
    dataset_widget,
    category_widget,
    count_widget
])

display(data_collection_widget)
```

Data Collection For Collision Avoidance

```
button_layout = widgets.Layout(width='128px', height='64px')
free_button = widgets.Button(description='add free', button_style='success', layout=button_layout)
blocked_button = widgets.Button(description='add blocked', button_style='danger', layout=button_layout)
free_count = widgets.IntText(layout=button_layout, value=len(os.listdir(free_dir)))
blocked_count = widgets.IntText(layout=button_layout, value=len(os.listdir(blocked_dir)))

from uuid import uuid1

def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_free():
    global free_dir, free_count
    save_snapshot(free_dir)
    free_count.value = len(os.listdir(free_dir))

def save_blocked():
    global blocked_dir, blocked_count
    save_snapshot(blocked_dir)
    blocked_count.value = len(os.listdir(blocked_dir))

# attach the callbacks, we use a 'lambda' function to ignore the
# parameter that the on_click event would provide to our function
# because we don't need it.
free_button.on_click(lambda x: save_free())
blocked_button.on_click(lambda x: save_blocked())

display(image)
display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([blocked_count, blocked_button]))
```

B.4 Training Model

Creating Model for Road following

```
import torch
import torchvision

device = torch.device('cuda')
output_dim = 2 * len(dataset.categories)  # x, y coordinate for each category

# RESNET 18
model = torchvision.models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, output_dim)

model = model.to(device)

model_save_button = ipywidgets.Button(description='save model')
model_load_button = ipywidgets.Button(description='load model')
model_path_widget = ipywidgets.Text(description='model path', value='road_following_model.pth')

def load_model(c):
    model.load_state_dict(torch.load(model_path_widget.value))
model_load_button.on_click(load_model)

def save_model(c):
    torch.save(model.state_dict(), model_path_widget.value)
model_save_button.on_click(save_model)

model_widget = ipywidgets.VBox([
    model_path_widget,
    ipywidgets.HBox([model_load_button, model_save_button])
])


display(model_widget)
```

```
BATCH_SIZE = 8
optimizer = torch.optim.Adam(model.parameters())

epochs_widget = ipywidgets.IntText(description='epochs', value=1)
eval_button = ipywidgets.Button(description='evaluate')
train_button = ipywidgets.Button(description='train')
loss_widget = ipywidgets.FloatText(description='loss')
progress_widget = ipywidgets.FloatProgress(min=0.0, max=1.0, description='progress')

def train_eval(is_training):
    global BATCH_SIZE, LEARNING_RATE, MOMENTUM, model, dataset, optimizer, eval_button, train_button, accuracy_widge

    try:
        train_loader = torch.utils.data.DataLoader(
            dataset,
            batch_size=BATCH_SIZE,
            shuffle=True
        )
        state_widget.value = 'stop'
        train_button.disabled = True
        eval_button.disabled = True
        time.sleep(1)

        if is_training:
            model = model.train()
        else:
            model = model.eval()

        while epochs_widget.value > 0:
            i = 0
            sum_loss = 0.0
            error_count = 0.0
            for images, category_idx, xy in iter(train_loader):
                images = images.to(device)
                xy = xy.to(device)
                if is_training:
                    optimizer.zero_grad()
```

```
                outputs = model(images)
                loss = 0.0
                for batch_idx, cat_idx in enumerate(list(category_idx.flatten())):
                    loss += torch.mean((outputs[batch_idx][2 * cat_idx:2 * cat_idx+2] - xy[batch_idx])**2)
                loss /= len(category_idx)

                if is_training:
                    loss.backward()
                    optimizer.step()

                count = len(category_idx.flatten())
                i += count
                sum_loss += float(loss)
                progress_widget.value = i / len(dataset)
                loss_widget.value = sum_loss / i

            if is_training:
                epochs_widget.value = epochs_widget.value - 1
            else:
                break
    except e:
        pass
    model = model.eval()

    train_button.disabled = False
    eval_button.disabled = False
    state_widget.value = 'live'

train_button.on_click(lambda c: train_eval(is_training=True))
eval_button.on_click(lambda c: train_eval(is_training=False))

train_eval_widget = ipywidgets.VBox([
    epochs_widget,
    progress_widget,
    loss_widget,
    ipywidgets.HBox([train_button, eval_button])
])

display(train_eval_widget)
```

Creating Model for Collision Avoidance

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms

dataset = datasets.ImageFolder(
    'dataset',
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)

train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - 50, 50])

train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0,
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0,
)

model = models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
```

```
NUM_EPOCHS = 25
BEST_MODEL_PATH = 'best_model_resnet18.pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

    test_error_count = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax(1))))

    test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
    print('%d: %f' % (epoch, test_accuracy))
    if test_accuracy > best_accuracy:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = test_accuracy
```

```
import torch
import torchvision

CATEGORIES = ['apex']

device = torch.device('cuda')
model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2 * len(CATEGORIES))
model = model.cuda().eval().half()
```

trt for Roadfollowing

```
model.load_state_dict(torch.load('road_following_model.pth'))
```

```
<All keys matched successfully>
```

```
from torch2trt import torch2trt
data = torch.zeros((1, 3, 224, 224)).cuda().half()
model_trt = torch2trt(model, [data], fp16_mode=True)
```

```
torch.save(model_trt.state_dict(), 'road_following_model_trt.pth')
```

trt for Collision Avoidance

```
model.load_state_dict(torch.load('best_model1_resnet18.pth'))
```

```
from torch2trt import torch2trt
data = torch.zeros((1, 3, 224, 224)).cuda().half()
model_trt = torch2trt(model, [data], fp16_mode=True)
```

```
torch.save(model_trt.state_dict(), 'best_model2_trt.pth')
```

B.5 Live Execution

Load the trained models

```
import torch
import torchvision
device = torch.device('cuda')
```

```
import torch
from torch2trt import TRTModule

model_trt = TRTModule()
model_trt.load_state_dict(torch.load('road_following_model_trt.pth')) # well trained road following model

model_trt_collision = TRTModule()
model_trt_collision.load_state_dict(torch.load('best_model_trt.pth')) # well trained collision avoidance model
```

```
<All keys matched successfully>
```

```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

```
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display
from jetcam.csi_camera import CSICamera
from jetcam.utils import bgr8_to_jpeg


camera = CSICamera(width=224, height=224, capture_fps=10)
camera.running = True

image = widgets.Image(format='jpeg', width=224, height=224)
blocked_slider = widgets.FloatSlider(description='blocked', min=0.0, max=1.0, orientation='vertical')

camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

display(widgets.HBox([image, blocked_slider]))
```

```
VBox(children=(HBox(children=(Image(value=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\x…
```

```
from jetracer.nvidia_racecar import NvidiaRacecar
car = NvidiaRacecar()
```

```
import torch.nn.functional as F
import time
import numpy as np

STEERING_GAIN = 0.8
STEERING_BIAS = -0.15

def update(change):
    global blocked_slider
    x = change['new']
    x = preprocess(x)
    y = model_trt_collision(x)
    y = F.softmax(y, dim=1)

    prob_blocked = float(y.flatten()[0])

    blocked_slider.value = prob_blocked
```

```
    if prob_blocked < 0.88:
        car.throttle = 0.170
        image = change['new']
        xy = model_trt(preprocess(image)).detach().float().cpu().numpy().flatten()
        z = xy[0]
        car.steering = z * STEERING_GAIN + STEERING_BIAS
    else:
        car.throttle = 0.0

update({'new': camera.value})

#alternate code loop
'''
while True:
    global blocked_slider
    image = camera.read()
    x = preprocess(image)
    y = model_trt_collision(x)
    y = F.softmax(y, dim=1)
    prob_blocked = float(y.flatten()[0])
    blocked_slider.value = prob_blocked

    if prob_blocked < 0.78:
        car.throttle = 0.158
        img = preprocess(image).half()
        output = model_trt(img).detach().cpu().numpy().flatten()
        z = float(output[0])
        car.steering = z * STEERING_GAIN + STEERING_BIAS
    else:
        car.throttle = 0.0'''
```

```
'\nwhile True:\n    global blocked_slider\n    image = camera.read()\n    x = preprocess(image)\n    y = model_trt
_collision(x)\n    y = F.softmax(y, dim=1)\n    prob_blocked = float(y.flatten()[0])\n    blocked_slider.value = p
rob_blocked\n    \n    if prob_blocked < 0.78:\n        car.throttle = 0.158\n        img = preprocess(image).half
()\n        output = model_trt(img).detach().cpu().numpy().flatten()\n        z = float(output[0])\n        car.st
eering = z * STEERING_GAIN + STEERING_BIAS\n    else:\n        car.throttle = 0.0'
```

```
camera.observe(update, names='value')
```

REFERENCES

[1] Al Malki, H. H., Moustafa, A. I., and Sinky, M. H. (2021). "An improving position method using extended kalman filter." *Procedia Computer Science*, 182, 28–37.

[2] Barba-Guaman, L., Eugenio Naranjo, J., and Ortiz, A. (2020). "Deep learning framework for vehicle and pedestrian detection in rural roads on an embedded gpu." *Electronics*, 9(4), 589.

[3] Burnett, K., Schimpe, A., Samavi, S., Gridseth, M., Liu, C. W., Li, Q., Kroeze, Z., and Schoellig, A. P. (2019). "Building a winning self-driving car in six months." *2019 International Conference on Robotics and Automation (ICRA)*, IEEE. 9583–9589.

[4] Caltagirone, L., Bellone, M., Svensson, L., and Wahde, M. (2019). "Lidar–camera fusion for road detection using fully convolutional neural networks." *Robotics and Autonomous Systems*, 111, 125–131.

[5] Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). "Multi-view 3d object detection network for autonomous driving." *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1907–1915.

[6] Daily, M., Medasani, S., Behringer, R., and Trivedi, M. (2017). "Self-driving cars." *Computer*, 50(12), 18–23.

[7] De Silva, V., Roche, J., and Kondoz, A. (2017). "Fusion of lidar and camera sensor data for environment sensing in driverless vehicles.

[8] Do, T.-D., Duong, M.-T., Dang, Q.-V., and Le, M.-H. (2018). "Real-time self-driving car navigation using deep neural network." *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, IEEE. 7–12.

[9] Fan, X., Zhang, Z., and Wang, Y. (2021). "Development of a self-driving car prototype for educational and research purposes." *Journal of Physics: Conference Series*, Vol. 1905, IOP Publishing. 012013.

[10] Febbo, R., Flood, B., Halloy, J., Lau, P., Wong, K., and Ayala, A. (2020). "Autonomous vehicle control using a deep neural network and jetson nano." *Practice and Experience in Advanced Research Computing*, 333–338.

[11] Feng, X., Jiang, Y., Yang, X., Du, M., and Li, X. (2019). "Computer vision algorithms and hardware implementations: A survey." *Integration*, 69, 309–320.

[12] Hecht, J. (2018). "Lidar for self-driving cars." *Optics and Photonics News*, 29(1), 26–33.

[13] Jain, A. K. (2018). "Working model of self-driving car using convolutional neural network, raspberry pi and arduino." *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, IEEE. 1630–1635.

[14] Jiang, Y., Gao, F., and Xu, G. (2010). "Computer vision-based multiple-lane detection on straight road and in a curve." *2010 International Conference on Image Analysis and Signal Processing*, IEEE. 114–117.

[15] Karni, U., Ramachandran, S. S., Sivaraman, K., and Veeraraghavan, A. (2019). "Development of autonomous downscaled model car using neural networks and machine learning." *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE. 1089–1094.

[16] Kim, J., Kim, J., Jang, G.-J., and Lee, M. (2017). "Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection." *Neural Networks*, 87, 109–121.

[17] Kim, J. and Park, C. (2017). "End-to-end ego lane estimation based on sequential transfer learning for self-driving cars." *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 30–38.

[18] Kumar, G. A., Lee, J. H., Hwang, J., Park, J., Youn, S. H., and Kwon, S. (2020). "Lidar and camera fusion approach for object distance estimation in self-driving vehicles." *Symmetry*, 12(2), 324.

[19] Li, J., He, X., and Li, J. (2015). "2d lidar and camera fusion in 3d modeling of indoor environment." *2015 National Aerospace and Electronics Conference (NAECON)*, IEEE. 379–383.

[20] Li, J., Zhang, X., Li, J., Liu, Y., and Wang, J. (2020). "Building and optimization of 3d semantic map based on lidar and camera fusion." *Neurocomputing*, 409, 394–407.

[21] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2020). "Deep learning for generic object detection: A survey." *International journal of computer vision*, 128(2), 261–318.

[22] Liu, Z., Cai, Y., Wang, H., Chen, L., Gao, H., Jia, Y., and Li, Y. (2021). "Robust target recognition and tracking of self-driving cars with radar and camera information fusion under severe weather conditions." *IEEE Transactions on Intelligent Transportation Systems*.

[23] Liu, Z., Yu, S., Wang, X., and Zheng, N. (2017). "Detecting drivable area for self-driving cars: An unsupervised approach." *arXiv preprint arXiv:1705.00451*.

[24] Maturana, D. and Scherer, S. (2015). "Voxnet: A 3d convolutional neural network for real-time object recognition." *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE. 922–928.

[25] Mulyanto, A., Borman, R., Prasetyawana, P., and Sumarudin, A. (2020). "2d lidar and camera fusion for object detection and object distance measurement of adas using robotic operating system (ros)." *JOIV Int. J. Informatics Vis*, 4(4), 231–236.

[26] Ramos, S., Gehrig, S., Pinggera, P., Franke, U., and Rother, C. (2017). "Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling." *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE. 1025–1032.

[27] Sell, R., Leier, M., Rassõlkin, A., and Ernits, J.-P. (2018). "Self-driving car iseauto for research and education." *2018 19th International Conference on Research and Education in Mechatronics (REM)*, IEEE. 111–116.

[28] Sharma, S., Tewolde, G., and Kwon, J. (2018). "Behavioral cloning for lateral motion control of autonomous vehicles using deep learning." *2018 IEEE International Conference on Electro/Information Technology (EIT)*, IEEE. 0228–0233.

[29] Tang, J., Li, S., and Liu, P. (2021). "A review of lane detection methods based on deep learning." *Pattern Recognition*, 111, 107623.

[30] Uddin, M. I., Alamgir, M. S., Rahman, M. M., Bhuiyan, M. S., and Moral, M. A. (2021). "Ai traffic control system based on deepstream and iot using nvidia jetson nano." *2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, IEEE. 115–119.

[31] Ullah, S. and Kim, D.-H. (2020). "Benchmarking jetson platform for 3d point-cloud and hyper-spectral image classification." *2020 IEEE International conference on big data and smart computing (BigComp)*, IEEE. 477–482.

[32] Vijitkunsawat, W. and Chantngarm, P. (2020). "comparison of machine learning algorithm's on self-driving car navigation using nvidia jetson nano." *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, IEEE. 201–204.

[33] Wang, Y., Goila, A., Shetty, R., Heydari, M., Desai, A., and Yang, H. (2017). "Obstacle avoidance strategy and implementation for unmanned ground vehicle using lidar." *SAE international journal of commercial vehicles*, 10(1), 50–56.

[34] Wang, Z., Ren, W., and Qiu, Q. (2018). "Lanenet: Real-time lane detection networks for autonomous driving." *arXiv preprint arXiv:1807.01726*.

[35] Yin, H. and Berger, C. (2017). "When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets." *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE. 1–8.

[36] Zaarane, A., Slimani, I., Al Okaishi, W., Atouf, I., and Hamdoun, A. (2020). "Distance measurement system for autonomous vehicles using stereo camera." *Array*, 5, 100016.

[37] Zhao, G., Xiao, X., Yuan, J., and Ng, G. W. (2014). "Fusion of 3d-lidar and camera data for scene parsing." *Journal of Visual Communication and Image Representation*, 25(1), 165–183.

[38] Zhao, J., Liang, B., and Chen, Q. (2018). "The key technology toward the self-driving car." *International Journal of Intelligent Unmanned Systems*.

[39] Zhao, X., Sun, P., Xu, Z., Min, H., and Yu, H. (2020). "Fusion of 3d lidar and camera data for object detection in autonomous vehicle applications." *IEEE Sensors Journal*, 20(9), 4901–4913.