DESIGN AND IMPLEMENTATION OF SENSING METHODS ON ONE-TENTH SCALE OF AN AUTONOMOUS RACE CAR

A Thesis

Submitted to the Faculty

of

Purdue University

by

Harshitha Veeramachaneni

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2020

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

I would like to dedicate this thesis work to my parents and my sister who have always supported and motivated me throughout my life and helped me face any challenge with courage and confidence.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my professor Dr. Lingxi Li for supporting me throughout my thesis with his valuable suggestions. His immense knowledge and patience have helped me in all the research work and also in writing my thesis.

Besides my advisor, I would sincerely like to acknowledge LHP Engineering Solutions for providing guidance and research resources for my thesis.

I would like to acknowledge my committee members Dr. Mohamed El-Sharkawy and Dr. Yaobin Chen for extending their support and challenging me with tough questions and projects during my course work.

I am very grateful to Dr. Brian King for extending me motivation and support for starting my master's degree at IUPUI.

I would like to thank Sherrie Tucker for her support during my entire master studies. Her timely remainders and notifications about all the necessary things made my departmental related works much easier.

Lastly, I would like to thank my parents and family for supporting me mentally and spiritually for being successful in life.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Veeramachaneni, Harshitha. M.S.E.C.E., Purdue University, December 2020. Design and Implementation of sensing methods on One-Tenth Scale of an Autonomous Race Car. Major Professor: Lingxi Li, Associate Professor.

Self-driving, is simply the capacity of a vehicle to drive itself without human intervention. To accomplish this, the vehicle utilizes mechanical and electronic parts, sensors, actuators and an AI computer. The on-board PC runs advanced programming, which permits the vehicle to see and comprehend its current circumstance dependent on sensor input, limit itself in that climate and plan the ideal course from point A to point B. Independent driving is not an easy task, and to create self-sufficient driving arrangements is an exceptionally significant ability in the present programming designing field.

ROS is a robust and versatile communication middle ware (framework) tailored and widely used for robotics applications. This thesis work intends to show how ROS could be used to create independent driving programming by investigating self-governing driving issues, looking at existing arrangements and building up a model vehicle utilizing ROS.

The main focus of this thesis is to develop and implement an one-tenth scale of an autonomous RACECAR equipped with Jetson Nano board as the on-board computer, PCA9685 as PWM driver, sensors, and a ROS based software architecture.

Finally, by following the methods presented in this thesis, it is conceivable to build an autonomous RACECAR that runs on ROS.

By following the means portrayed in this theory, it is conceivable to build up a self-governing.

1. INTRODUCTION

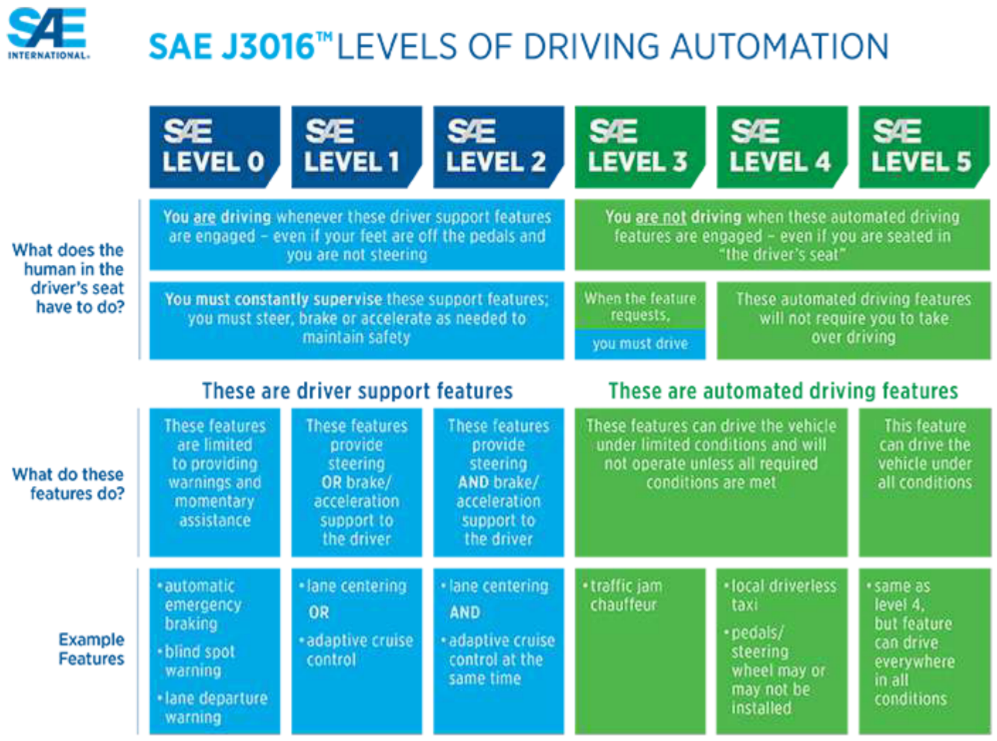
Throughout these last few years, the automotive industry has improved enormously in different areas, with the invention of novel and advanced technologies, regardless of whether it's in telematics, CAN communications, battery management systems, self-driving cars or driving assistance systems [1].

Self-driving cars also called as autonomous cars have been glorified and envisioned as the upcoming navigation system in urban areas where the travelers will be taken to their destination, without even applying any force on the accelerator, brake nor directing a wheel [2].

Now a days, autonomous driving isn't a dream nor an irrational thought, as large number of self-driving vehicles have started its operations in urban areas throughout the globe, with extraordinary achievement. Nevertheless, these frameworks have huge complications, with a large amount of work still needs to be accomplished and they need to attain the full attention to different maneuvers that occurs around it. Therefore, all significant vehicle manufacturers are initiating a step by step reconciliation of ADAS (Advanced Driver Assistance Systems) tasks that will empower autonomous driving in a more advanced way.

According to the standards of SAE (Society of Automotive Engineers), there are six levels of autonomous driving from Level 0 to Level 5. These levels choose how independent a vehicle is and the necessities and safety measures for a manual driver [1]. Level 0, No automation. This level insists that none of the autonomous features are available, where driving just depends in human maneuvers in each circumstance and the framework just gives warnings or passing help [1].

Level 1, Driver assistance. This framework is answerable for helping manual driver with a particular assignment. Cruise control, Lane Keep Assistance (LKA) are some



SAE J3016™ LEVELS OF DRIVING AUTOMATION

	SAE LEVEL 0	SAE LEVEL 1	SAE LEVEL 2	SAE LEVEL 3	SAE LEVEL 4	SAE LEVEL 5
What does the human in the driver's seat have to do?	You **are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			You **are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Fig. 1.1. SAE (0 - 5) levels of Driving Automation [3]

of the examples of this level since, the framework is fit for assuming responsibility for an activity, such as steering or slowing down [1].

Level 2, Partial automation. This framework can achieve at least two programmed errands, that work close by to help driver, anyway it is not yet viewed as an autonomous vehicle and the driver is needed to keep up the vehicle leveled out and take the fundamental actions if necessary. The vehicle performs activities, for example, steering, accelerating and decelerating simultaneously [1].

Level 3, Conditional automation. From this stage of autonomy, vehicle takes the capability of driving. The framework is answerable for the ordinary activities in driving and just requires the driver to take necessary measures, if the framework needs help or an emergency situation shows up. That being stated, in spite of the fact that the vehicle is generally driven automatically, the driver should at alert mode during high traffic and be prepared to take control when the framework requests [1].

Level 4, High automation. From this stage, the car is able monitor the climate and works in complete autonomous mode for a wide range of driving situations. Additionally, on contrary to level3, the framework is capable for protecting its travelers from accidents, regardless of whether it requires the driver intervention and the driver doesn't appropriately intervenes in time. In a situation that the structure finds that, not all conditions are met, at that point the driver must take control, until the vehicle rematches the autonomous capability [1].

Level 5, Full automation. In this level, the vehicle is fully autonomous where the car can drive in any scenario. It is as good as a human driver. All the emergency situations are handled by itself without any human intervention. The fully autonomous capabilities in level 5 raises many concerns which are to be considered during the development of vehicle architecture [1].

Autonomous driving is not one single innovation, yet rather a profoundly unpredictable framework that comprises of many sub-frameworks. It can be mainly classified into three parts. First one being algorithm development, which includes sensing, perception, and decision making. Second, the client systems, which includes operating system and hardware platform. Final one being the cloud platform, which includes deep learning model training, HD (high-definition) map, simulation and data storage [4].

In algorithm development stage the main task is to separate the significant data from sensor raw information, to comprehend its current circumstance and to settle on choices about its future activities. The client systems coordinate these algorithms together to meet ongoing and unwavering quality necessities. For example, let us consider we are using a camera that runs at 60 FPS, the client system needs to ensure that the longest phase of the preparing pipeline takes under 16 ms to finish. The cloud platform gives offline processing, and capacity abilities for self-driving vehicles. With the cloud stage, we can test new algorithms, update HD maps, and train for faster recognition, tracking, and decision models [4].

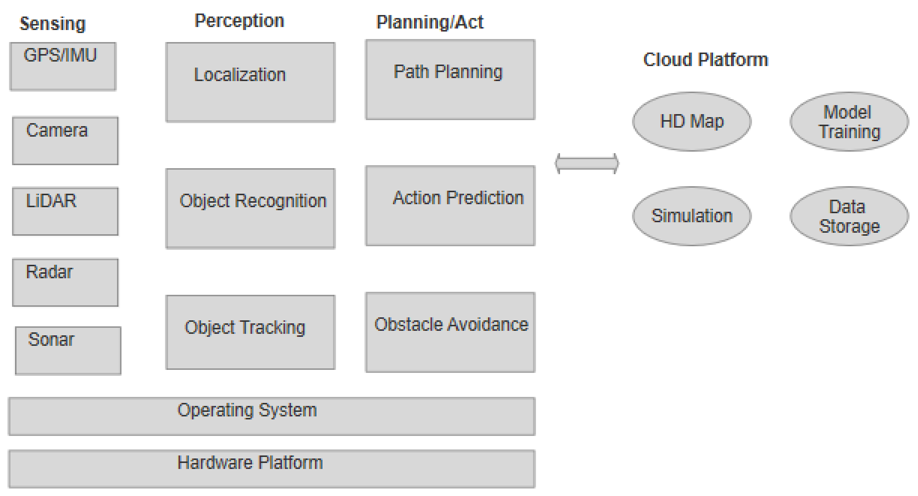


Fig. 1.2. Autonomous Driving Architecture [4]

1.1 Autonomous Driving Algorithms

The algorithm development stage mainly consists three main parts: Sensing, that is extracting important data from sensor raw information; perception, which is to know the its current position, orientation and to know its current environment; and decision, which is to plan its route and act according to its plan to reach their destinations safe [4].

1.1.1 Sensing

Typically, an autonomous vehicle comprises of many sensors. As each sensor has its own advantages and disadvantages, in autonomous cars, the information from different sensors must be consolidated for reliability and security [4].

GPS/IMU

The GPS/IMU framework enables the autonomous car to limit itself by detailing both inertial updates and a worldwide position estimates at a higher rates. GPS gives us accurate localization values, however its update rate is slow, and subsequently not suitable for giving constant updates. However, IMU errors aggregate after some time, resulting the errors in the position estimates. Nevertheless, an IMU has higher update rates, which fulfills the real time requirements. Therefore, by using both GPS and IMU, we can give precise and real time updates for an autonomous car localization [4].

LiDAR

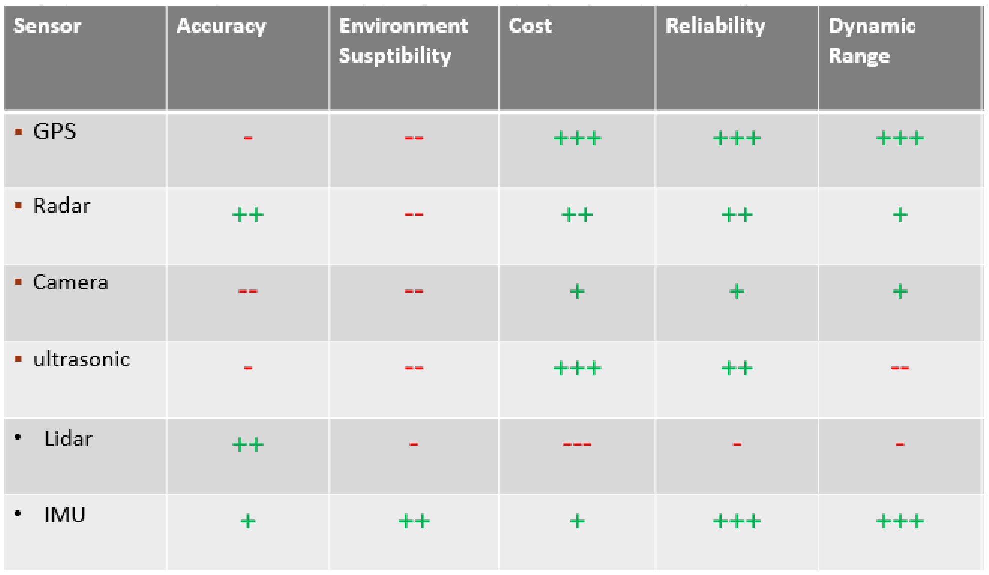
LiDAR is the most promising sensor which is used for mapping, localization and obstacle avoidance. The working principle of LiDAR that it creates Laser beam train, which beam off the surface and calculates the time it takes to return to its source. It has a high accuracy, which helps in producing HD maps, to localize a moving car against HD maps, to detect obstacles in-front of that and so on [4].

Cameras

The most complicated task in the entire concept of autonomous driving is object detection and object tracking. Cameras does an excellent job in these tasks. Lane Detection, Caution Sign detection, pedestrian detection are some of such examples. To upgrade autonomous vehicle security, existing executions use at least eight 1080p cameras around the vehicle, with the end goal that we can utilize cameras to distinguish, perceive, and track obstacles before, behind, and on the two sides of vehicle [4].

Radar and Sonar

The radar and sonar framework is generally utilized for short-range detection of obstacles. Radar is extremely useful in bad weather conditions. The information produced by radar and sonar does not need a lot of computational power. These are used in the applications such as braking, swerving and so on [4].



Sensor	Accuracy	Environment Susptibility	Cost	Reliability	Dynamic Range
▪ GPS	-	--	+++	+++	+++
▪ Radar	++	--	++	++	+
▪ Camera	--	--	+	+	+
▪ ultrasonic	-	--	+++	++	--
• Lidar	++	-	---	-	-
• IMU	+	++	+	+++	+++

Fig. 1.3. Merits and Demerits of Different Sensors

1.1.2 Perception

Perception is using sensed and fused data for meaningful interpretation of the environment at every moment in time. Localization, object detection, and object tracking are the three main tasks in perception. It worries in defining an estimate of its general surroundings with the end goal that choice can be made in regards to control of the vehicle. This may incorporate what static hazards or obstructions exist for route of the way ahead. On the off chance that dynamic objects exist and the estimate of its future position meets the autonomous way. Hence, the perception provides self and situational awareness of its surroundings [4].

1.1.3 Plan

Commonly in the route plan, the general mission of a autonomous vehicle is to go from a start position to a destination position. During the route plan objective, numerous choices are to be made about a short range plan (trajectory) which includes questions like:

- When to start a turn?
- How to avoid slow moving traffic?
- What lane to be in?
- What is the best course to avoid and obstacle or an impending collisions with another moving object?

All of the sensor, perception and situational awareness leads to determination of the short-range plan (The trajectory). In the short range (time), the path plan defines the position and velocity that the vehicle should be in at each instant in time (sample time). The path plan is not static and will be modified at each algorithm sample time as new information is obtained via the perception algorithms.

1.1.4 Act

Act is the process of execution of the plan. The path plan is executed by the vehicle control algorithms. The controller uses the vehicle actuators (brakes and accelerators, steering effects) to meet to path plan trajectory profile. The actual path the vehicle takes is a function of the vehicle performance (the system) and the controller.

1.2 Autonomous Driving Client System

The client system incorporate the previously mentioned calculations together to meet real-time and dependability prerequisites. The following are some of the challenges that occurs during this process: the framework needs to guarantee that the handling pipeline is sufficiently quick to deal with the large amount of sensor information produced; if a part of the framework fails or behaves abnormally, it must be adequately vigorous to recover from the failure and, moreover, all the computations needs to be done within the available space and resources [4].

1.2.1 Robot Operating System (ROS)

ROS is a robust and versatile communication middle ware (framework) tailored and widely used for robotics applications. It is a powerful operating system for autonomous driving aside from the fact that it experiences a few issues such as reliability, performance and security. More about ROS is discussed in the future chapters [4].

1.2.2 Hardware Platform

Hardware Platform is one of the most important factor for operating an autonomous car in real time scenarios. To perform these tasks a powerful computing system is required to interpret a lot of detecting information and to execute complex perception algorithms. The various computing devices available in the market are personal computers (PC), real-time embedded computer, multicore CPU servers, Graphics processing units (GPU) and Field-programmable gate arrays (FPGA). We can choose any one of the mentioned computing systems based on objectives, performance, reliability and cost [5].

1.3 Autonomous Driving Cloud Platform

Autonomous vehicles are versatile frameworks, and consequently they need a cloud based platform to provide support. Distributed computing and distributed storage are two main aspects of cloud platform. Simulation, HD map production, and deep learning model training are some of its several applications [4].

Therefore, the above mentioned pipeline for autonomous driving constitutes a level 5 autonomous vehicles. However, the issue is, that driving with level 5 vehicles isn't allowed on open streets, the test vehicles are costly and require a great deal of support and the recording of information takes a very long time. Moreover, as a college research student it is especially hard to get the time, cash and equipment to test autonomous driving capabilities in certifiable situations. To bridge such disadvantage, considerations were placed into little, minimal effort platforms, that help the usage of similar methodologies replicated in real size self-driving vehicles.

As a significant objective of enthusiasm for the research network, this quickens the interest to accomplish more faster and efficient results, with the serious perspective among research scientists, university students and engineers.

1.4 Context

In the part of automated and mechanical applications for autonomous vehicles, this work was created and developed at LHP Engineering Solutions.

This platform provides an self-driving car, that permits the development, testing and approval of various algorithms, from simple distance calculation to a computer vision techniques, LiDAR scan and so on that implements the of scenarios Advanced Driver Assistance Systems (ADAS).

1.5 Research Objectives

The significant goal of this work is to construct and build up an automated racecar to develop various real time scenarios such as collision avoidance, blind spot detection and lane changing algorithms. The main motive, is to build up a racecar, to develop, test and approve different tools, with respect to ADAS functionalities and the other parts such as chassis, servos and so on which are used in autonomous driving.

1.6 Thesis Contributions

The primary commitments and contributions of this work are:

- Assembling and modifying the Traxxas remote control car into an autonomous car
- Analyzing the raw data from different sensors like Camera, LiDAR and Radar

- Implementing and running of RACECAR, which will permit to test and validate different sensor technologies
- Implementing and evaluating different use case scenarios like Lane Following, Obstacle Avoidance, Lane Keep Assist (LKA) and so on

1.7 Thesis Structure

The structure from Chapter 2 is as follows:

- Chapter 2: In this chapter, all the technologies were listed with their necessary software installations.
- Chapter 3: In this chapter, both hardware and software architecture was described.
- Chapter 4: Here, all the algorithms that were implemented on the RACECAR were listed.
- Chapter 5: Finally, all the results and conclusions were made.
- Chapter 6: This chapter is about the future work and further improvements.

2. SMALL SCALE AUTONOMOUS VEHICLE PLATFORMS

These small scale platforms give scientists and university students a bench setup to mimic and mimic different ways to deal with the aspect of self-driving, by utilizing one-tenth size of an actual car. Versatile to various instance of research, this RACECAR gives a reduced work tools, demonstrating the best way to assemble a vehicle, stating the necessary hardware equipment and the software installations required to setup the Robot Operative System (ROS) and other required software [1].

By giving a easy and simple test bed, it empowers the user to concentrate on different methodologies of algorithm development along with examining and verifying. Utilizing its features, there is a chance to put more efforts on perception and decision making steps of autonomous driving stack, by utilizing the sensors like ultrasonic, radars, cameras and LiDAR [1].

Different technologies like V2V and V2X can be explored, examining situations like moving towards an roundabout or T junction or a four way junction, that have been intensely explored in the most recent years.

On focusing these thesis objectives, different sensor technologies have been explored and implemented for different use case scenarios.

2.1 Hardware Assembly of RACE CAR

The hardware assembly of a RACE CAR can broadly be divided into two parts. The first part is the Chassis Preparation and the second part is Platform Preparation.

2.1.1 Chassis Preparation

The RACECAR Chassis is based on the TRAXXAS Slash 4 X 4 Platinum truck, an upgraded version of the normal Slash 4 X 4. The frame is available directly and it can be used as it is. Basically, remove the pieces that are not required, update the springs, and include another front guard. Here, we modified the chassis of the Traxxis to support the extra weight from electronics so that it can be converted into autonomous mode [6]. Fig.2.1. shows the modified version of TRAXXAS car that fits the needs of our autonomous RACECAR.

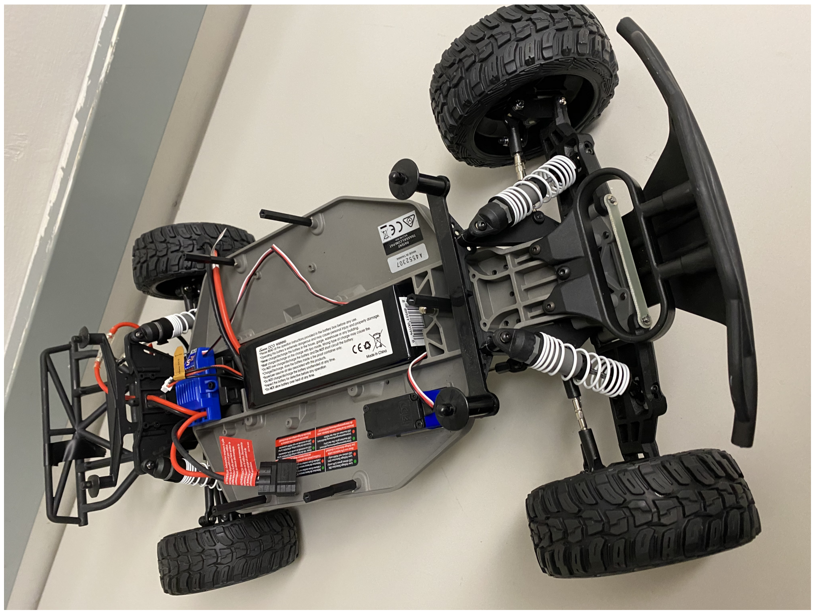


Fig. 2.1. CHASSIS

2.1.2 Platform Preparation

Platform preparation comprises of two sections. The initial segment is the frame, a one-tenth scale controller vehicle with changes to help the additional load of hardware. The subsequent part is the electronics. The hardware comprises of the AI computer and sensors, which empower our car to act autonomously [7].

Frame

The hardware is attached to platform decks, which on account of RACECAR are produced using exactness cut 3/16 Delrin sheets [7].

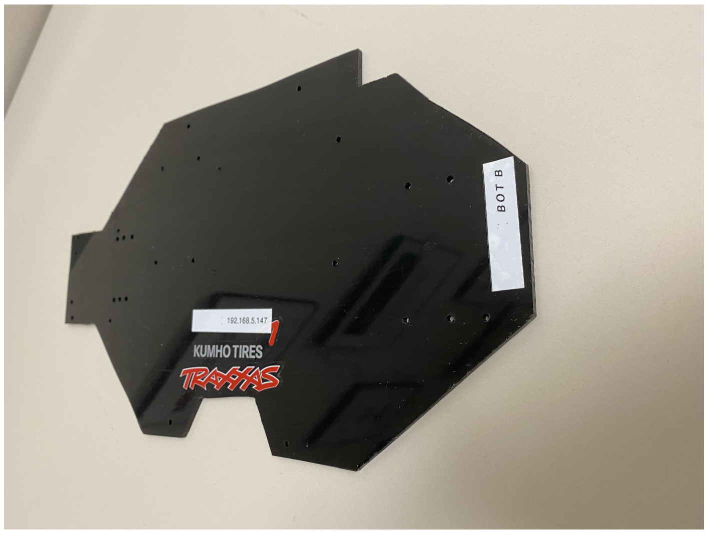


Fig. 2.2. PLATFORM DECK

Jetson NANO

NVIDIA® Jetson Nano^{TM} Developer Kit is a little, amazing computer that runs a numerous neural systems in equal for applications like mage classification, object detection, segmentation, and speech processing. All in a simple to-utilize stage that can run in a very low power as 5 watts [8]. It is the on-board computer on my Race car. The Jetson NANO is a low cost, Ubuntu 16.04 based electronic control unit(ECU) with a separate GPU. This ECU is extremely reliable for students and researchers. A short summary of Jetson NANO Developer Kit Technical Specifications is listed in the Table. 2.1.

Fig. 2.3. Jetson NANO [8]

Table 2.1.
Jetson Nano Specifications [8]

Ubuntu	16.04
GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Video Encoder	4K @ 30 — 4x 1080p @ 30 — 9x 720p @ 30 (H.264/H.265)
Video Decoder	4K @ 60 — 2x 4K @ 30 — 8x 1080p @ 30 — 18x 720p @ 30— (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and DP
USB	4x USB 3.0, USB 2.0 Micro-B
Interfaces	GPIO, I2C, I2S, SPI, UART

Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685

An easy and simple method to control Servo Motors on the NVIDIA Jetson Nano Developer Kit is to associate the Jetson Nano and PWM driver through I2C communication protocol. We have used Adafruit 16-Channel 12-bit PWM/Servo Driver On our Race car. The greatest advantage of this micro controller is that, just by utilizing two pins, we can control 16 free-running PWM outputs. It even allows to fasten up 62 breakouts to control up to 992 PWM outputs. The PCA9685 breakout board is a decent application to explicit the miniaturized scale regulator for servo control [9].

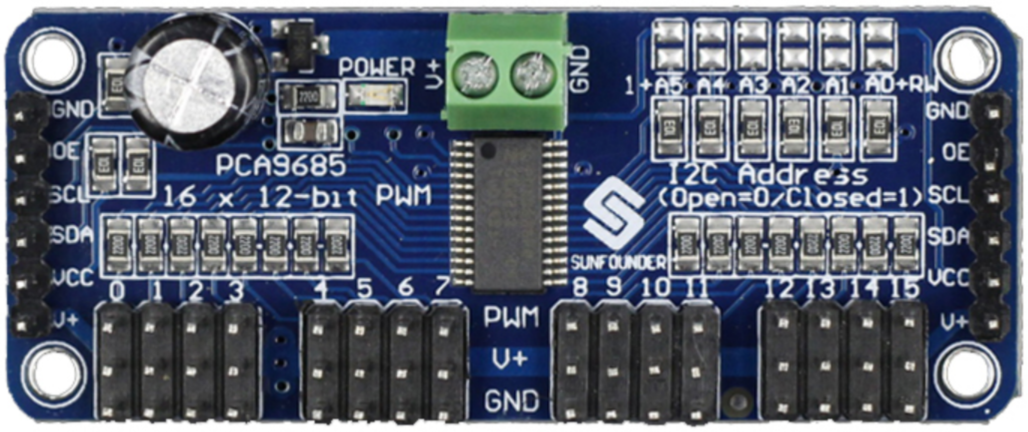


Fig. 2.4. PCA9685 [9]

Buck Converter

A buck converter is used to convert the voltage from LPIO battery (11.1v) to Jetson Nano(5v). It is a step-down DC-to-DC power converter which steps down voltage from its input (supply) to its output (load). SING F LTD Buck Converter DC to DC 5A 75W Input 4-36V Output 1 25-36V Step Down Regulator Power Module is the converter that we used on our RACECAR [10].

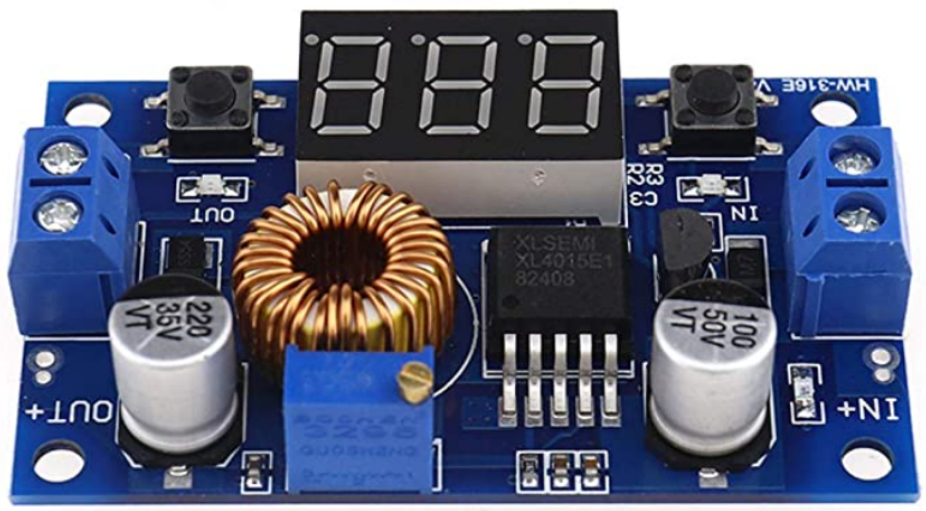


Fig. 2.5. Buck Converter [10]

Ultrasonic Sensor: HC-SR04

Ultrasonic distance sensor gives steady and exact distance estimations from 2cm to 450cm. It has a focal point of under 15 degrees and a precision of about 2mm [11]. A technical specification of HC-SR04 is as follows:

Table 2.2.
HC-SR04 Specifications [11]

Effectual Angle	less than 15 degrees
Ranging Distance	2cm – 400cm
Measuring Angle	30 degree
Power Supply	+5V DC

This sensor utilizes ultrasonic sound to measure the distance similar to bats and dolphins. Ultrasonic sound has such a high pitch, that people cannot hear it. This specific sensor conveys an ultrasonic sound that has a recurrence of around 40 kHz. The sensor has two fundamental parts: a transducer that makes an ultrasonic sound and another that tunes in for its reverberation. To utilize this sensor to measure separation, the micro controller should calculate the time it takes for the ultrasonic sound to travel [11]. Sound goes at approximately 340 meters for each second. This compares to about 29.412μs (microseconds) per centimeter. To measure the distance the sound has travelled we utilize the formulae: Distance is equal to $(Time * SpeedOfSound)/2$. First the sound travels from the sensor, and afterward it skips off of a surface and returns back [11].

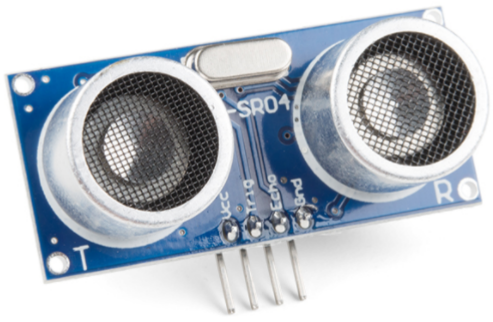


Fig. 2.6. Ultrasonic Sensors [11]

RPLiDAR A1

RPLIDAR A1 depends on laser triangulation extending standard and uses fast vision obtaining and handling equipment. The rate at which it measures the distance range is more than 8000 times per second. It runs clockwise to perform a 360 degree omni directional laser range scanning of its surrounding environment and then generate an outline map for the environment. We can adjust the scan rate from 2 - 10 Hz and it is ideal device for robot navigation and localization [12].

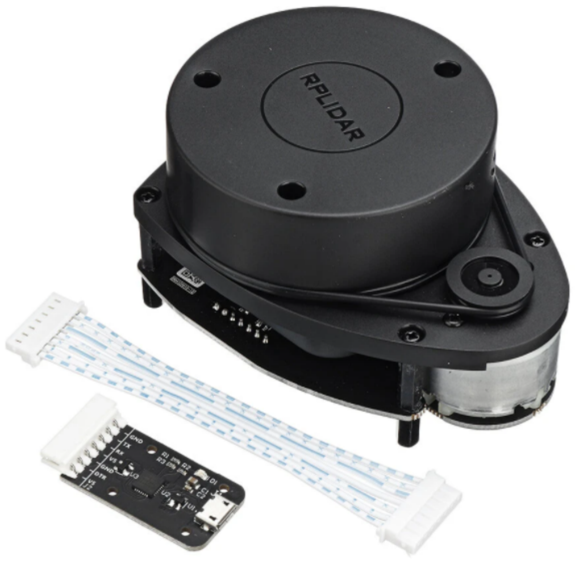


Fig. 2.7. RP LiDAR A1 [12]

Logitech C270 Desktop Webcam - USB Camera

This USB camera works at 30 frames per second with smooth video quality. Even the images are crisp, bright and contrasted [13].

The technical specifications of Logitech C270 webcam is as follows:

Table 2.3.
Camera Specifications [13]

Resolution	720p/30fps
Video Quality	Good
Focus	Fixed
Lens	Standard
Field of View	60
Microphone	Mono

Fig. 2.8. USB Camera [13]

LiPo Batteries

LiPo battery packs are made out of 3.7-volt cells. The cells are stacked and encased in an intense, semi-inflexible wrap. Traxxas LiPo batteries are offered in 2 cell (7.4 volts), 3-cell (11.1v), and 4 cell (14.8v) arrangements. LiPo batteries have more prominent "vitality thickness" than NiMH batteries, which means they have more voltage and limit than a NiMH battery would of a similar volume. LiPo batteries additionally support a higher voltage for a more noteworthy term of each run, so your model will have more "punch," speed up, and run longer with a LiPo than it would with a practically identical NiMH pack [14].

Fig. 2.9. LiPO Batteries [14]

Perceptin SSR V2.0 Radar

It is an industrial grade Millimeter-Wave Radar [15]. Following points are to be noted while installing the RADAR:

- Install vertically and make sure the interface is facing down
- Installation height (recommended): 50-100cm
- Pitch angle will lead to the detecting distance reduction
- Frames are returned every 50ms (20Hz)
- Maximum 64 objects are detected per frame
- Maximum Range of detection is 70m

We need an extra hardware equipment such as Kvaser CAN leaf, CAN breakout box and an extra power source to decode and run the perceptin radar. The technical specification of perceptin radar is shown in the Table 2.4.

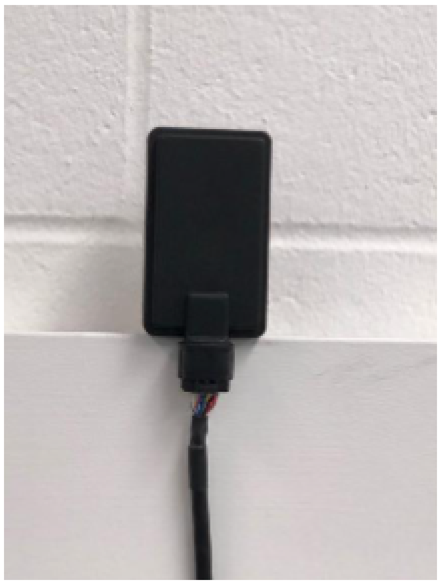


Fig. 2.10. Perceptin SSR V2.0 Radar [15]

Table 2.4.
PerceptIn Radar Specifications

Specs	**Unit**	**Range**
Azimuth Angle	Degree	60 ±1
Elevation Angle	Degree	±15
Angle Resolution	Degree	1
Maximum Range	Meter (m)	70(±0.16)
Installation Height	centimeters (cm)	50 to 100
Speed Range	m/s	-50 ∼ 15
Update Interval	ms	20 (20 Hz)
DC Voltage	Volts (v)	9 ∼ 36
Size	mm^3	102.5 x 52 x 28.5
Temperature	Celsius	40 ∼ 85
Output Interface	-	Double CAN with 500 kbps
Connector	-	Sumitoma 61897608

2.2 Initial Assembly

In the above two sections, chassis and platform decks were built. Finally, both of them were combined to complete the entire assembly. Here, the hardware is selected according to the build configuration. We modified this section by choosing the specific hardware required for our specific applications. For example, in Lane Following racecar, we mounted camera sensor as shown in the Fig. 2.11. Similarly, for LiDAR based racecar camera is replaced with LiDAR. Moreover, if we are building a fusion based racecar, we will use multiple sensors like lidar, camera and radar on the same build.

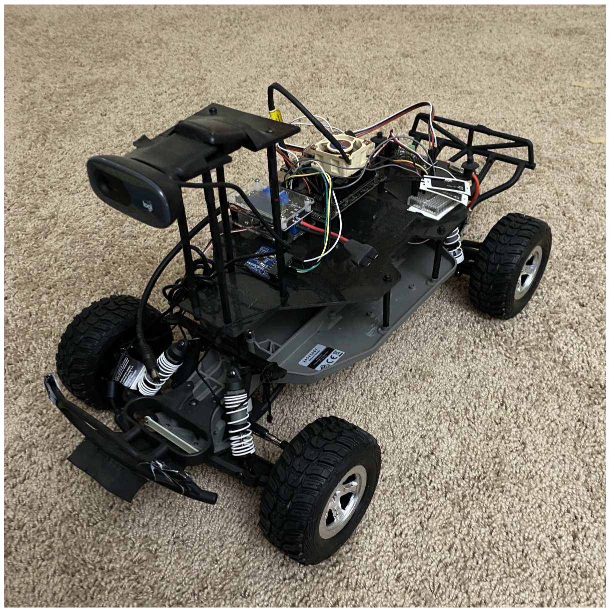


Fig. 2.11. Race Car

3. TECHNOLOGIES AND SOFTWARE INSTALLATIONS

3.1 Getting Started Jetson Nano

Things required before booting NVIDIA Jetson Nano

- micro-SD card (32 GB)
- 5V 2.5A MicroUSB power supply. I used Adafruit's 5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable (GEO151UB-6025) for my NANO.
- Ehernet cable

3.1.1 Downloading and Flashing the .img file in my 32GB micro-SD card

First, I downloaded the Jetson Nano Developer Kit SD Card Image from NVIDIA'S website [8]. For my windows operating system, I followed all the necessary steps provided in NVIDIA website [8] that are required for flashing .img file to my 32GB micro SD card.

3.1.2 Setup and First boot

After downloading and flashing the .img file to my micro-SD card, I inserted the card into the micro-SD slot on Jetson Nano. Next step is to connect the power supply and boot our module. Finally, by connecting Jetson Nano to a HDMI, output screen is displayed as follows.

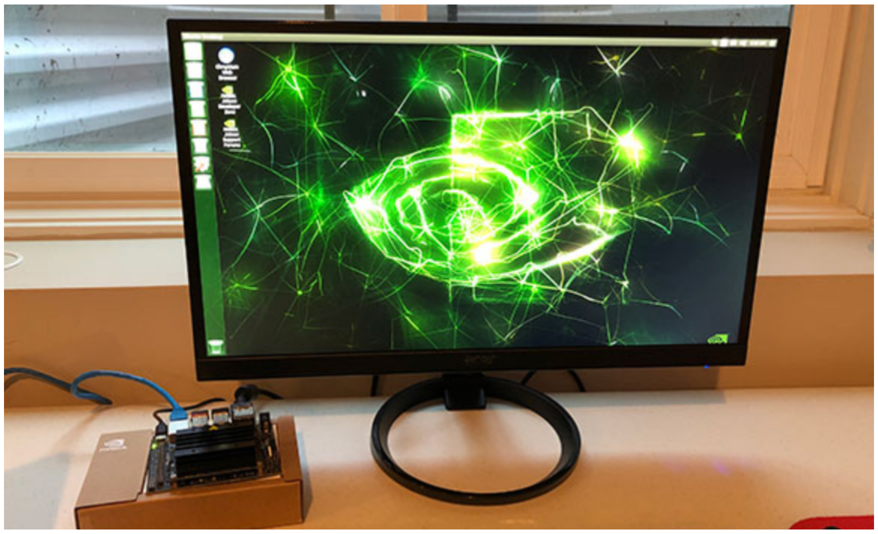


Fig. 3.1. Output screen of NANO after the Initial Boot

3.2 Robotic Operating System (ROS)

The Robot Operating System (ROS) structure, kept up by Willow Garage and Open Source Robotics Foundation (OSRF) since 2007, is an publicly available middle ware which experienced quick turn of events and has been generally used to plan mechanical autonomy applications. With its numerous product systems it gives an assortment of instruments, libraries and shows that it encourages the formation of mechanical aspects, boosts its code reuse-ability, and critical thinking through the automated network. In spite of the fact that ROS is certifiably not a constant structure, it is conceivable to incorporate it with continuous code [16].

ROS working principle is based on a procedure, which communicates through Nodes that empowers executable to be independently structured at run time. Every node is capable with a single task and correspondence between the nodes supports a distribute/supporter model, therefore giving an extremely straightforward and clean method of associating different software and hardware parts [17].

As explained in [18] ROS has four degrees of ideas, separated as it follows: ROS File System Level, ROS Computational Level and ROS Community Level.

The general assets of ROS are covered by File System Level which incorporates Packages, Services and Messages. For arranging the programming structure, the principle unit in this framework is Package. It constitutes of ROS run-time forms, ROS-subordinate library, data sets and so on. Packages are the most nuclear form thing and delivery system in ROS. Implying that the most coarse thing that can be manufactured, delivered is package. Messages portrayal has messages utilized in every package. Service depiction describes solicitation, reaction information for structuring the ROS services.

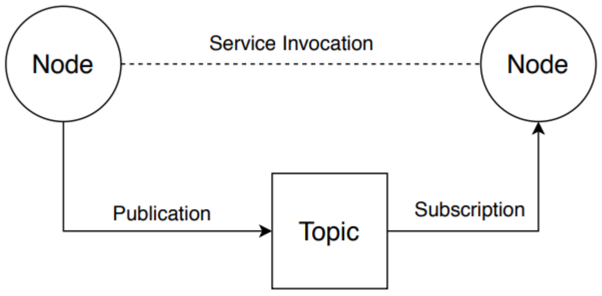


Fig. 3.2. Graphical Representation of Publisher and Subscriber [16]

The Computation Graph level is the thing that sets up the correspondence among ROS forms that prepares information all together. Crucial ideas to be considered are:

- Master [19] - ROS Master gives naming along with enlistment services to all other nodes in its framework. It hunts down publishers and subscribers of all its available topics and services. Its main task is to empower each node of ROS in finding each other. When these nodes have found each other, they speak with one another distributed.

- Nodes [20] - The main task of Nodes is to do calculations. Robot framework generally include numerous nodes. Two nodes in a framework can communicate with each other.

- Messages [21] - A message is an information structure, containing of composed fields. Nodes uses messages to pass data among one another.
- Topics [22] - Messages should be moved through a defined course. To do that Topics are responsible for directing the messages by means of a framework with the semantics of both publisher and subscriber. Two nodes communicate with each other via topic. It is used in recognizing the message substance. A node which depends on particular sort of information has to subscribe to a related topic. A solitary topic can have a numerous simultaneous publishers and subscribers, and a specific node may publish as well as subscribe to different topics. When all the process is said to be done, publishers and subscribers do not know about every others' presence. The thought is to decouple the creation of data from its utilization.

Community level, which is final level in ROS is liable for distributing and giving assets, information passing, programming with particular networks. It also gives repositories, forums and also question and answer site which addresses committed questions of ROS. In my research work, I have used the ROS Melodic distribution.

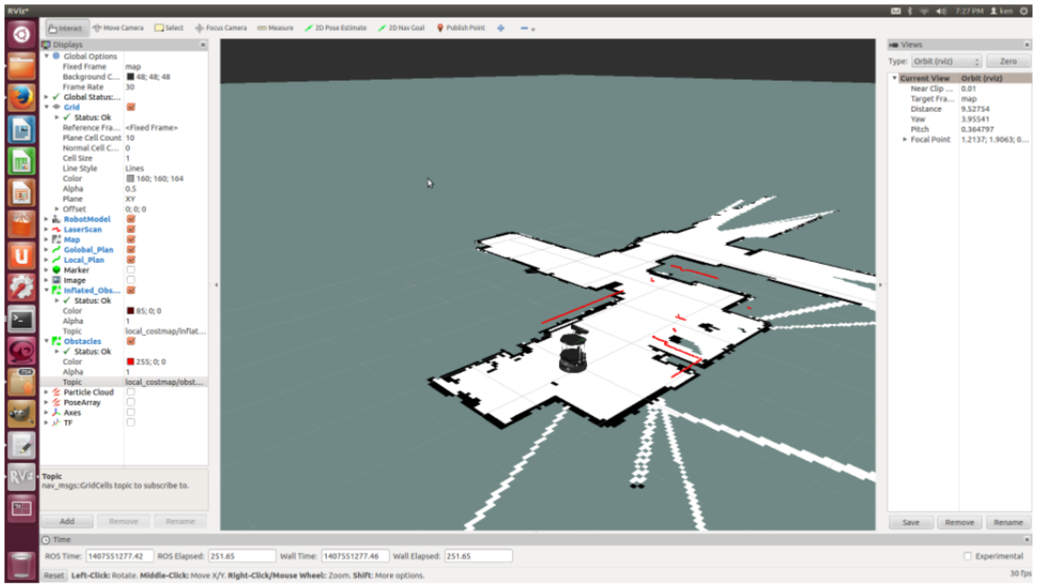
Remembered for the broad range of applications ROS gives two important applications in the advancement of automated and robotic field, being ROS visualization, Rviz and ROS rqt graph.

3.2.1 Rviz

Rviz is a three-dimensional visualizer used to envision robots, the situations they work in, and sensor information. It is a profoundly configurable application, with a wide range of sorts of plugins and modules [23].

As to sensor visualization, Rviz permits to show 3D and 2D sensors data from stereo cameras, lasers, kinetics, webcams, and so on.

Fig. 3.3. depicts the an example of a rivz window, which in turn gives the current configuration of a virtual bot.



```
Global Options
    Fixed Frame          map
    Background C...      48; 48; 48
    Frame Rate           30
  Global Status:...
Grid                     ✓
    Status: Ok
    Reference Fra...     <Fixed Frame>
    Plane Cell Count     10
    Normal Cell C...     0
    Cell Size            1
    Line Style           Lines
    Color                160; 160; 164
    Alpha                0.5
    Plane                XY
    Offset               0; 0; 0
RobotModel               ✓
LaserScan                ✓
Map                      ✓
Golobal_Plan             ✓
Local_Plan               ✓
Marker                   ☐
Image                    ☐
Inflated_Obs...          ✓
    Status: Ok
    Color                85; 0; 0
    Alpha                1
    Topic                local_costmap/inflat...
Obstacles                ✓
    Status: Ok
    Color                255; 0; 0
    Alpha                1
    Topic                local_costmap/obst...
Particle Cloud           ☐
PoseArray                ☐
Axes                     ☐
TF                       ☐

Topic
nav_msgs::GridCells topic to subscribe to.
```

```
Type: Orbit (rviz)
Current View     Orbit (rviz)
  Near Clip ...  0.01
  Target Fra...  map
  Distance       9.52754
  Yaw            3.95541
  Pitch          0.364797
  Focal Point    1.2137; 1.9063; 0...
```

```
Time
ROS Time: 1407551277.42   ROS Elapsed: 251.65   Wall Time: 1407551277.46   Wall Elapsed: 251.65   ☐ Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.   30 fps
```

Fig. 3.3. Rviz Window [23]

3.2.2 rqt Graph

rqt graph gives a GUI module to see the ROS computation graph. Its segments are made nonexclusive with the goal that different packages where you need to accomplish graph portrayal can rely on this particular package [24].

3.3 Jetson GPIO Library Package

The development boards from Jetson like Nano, TX1 etc comprises of forty-pin GPIO header. These GPIOs can be controlled for computerized information and yield utilizing the Python library mentioned in the Jetson GPIO Library package.

I referred to [25] for package installation into my Jetson Nano board.

3.4 Adafruit Servokit Library

I referred [26] to clone servokit's repository. These packages are necessary to make servo engines working with the NVIDIA Jetson Nano Developer Kit utilizing a PCA9685 breakout board over I2C communication protocol.

3.5 OpenCV

OpenCV is a library of programming functions, which is created to help applications that utilizes real time computer vision applications. It gives many capabilities to record, examine, and control of visual information and can wipe out a portion of the problem that software engineers face while creating applications that depend on computer vision. OpenCV has been utilized in both useful and innovative applications including self-directing vehicles and new types of computerized craftsmanship [27].

I referred to [28] for installation of OpenCV package on my Jetson Nano.

3.6 USB Camera Software Installations

I referred [29] to clone the github repository, which contains necessary ros drivers and packages for using Logitech camera.

3.7 RP LiDAR software Installations

I referred [30] to clone the github repository, which contains necessary ros drivers and packages for using RP LiDAR A1.

3.8 Perceptin SSR V2.0 Radar Software Installations

I referred [31] to clone the github repository, which contains necessary ros drivers and packages for using Perceptin Radar.

4. SYSTEM ARCHITECTURE

This section portrays the system architecture, describing how every tool and component are associated with one another, resulting in a final autonomous racecar.

4.1 Hardware Architecture

The racecar follows the hardware architecture as shown in the Fig. 4.1. Each and every part on the racecar is associated with the on board system, the Nano, that takes care of all the required computations. The connection between Camera and Lidar is established through USB, PWM driver is via I2C communication protocol and Ultrasonic sensors via GPIO ports. To supply power to the racecar, Lipo batteries were used to power up the servo and for DC motor.

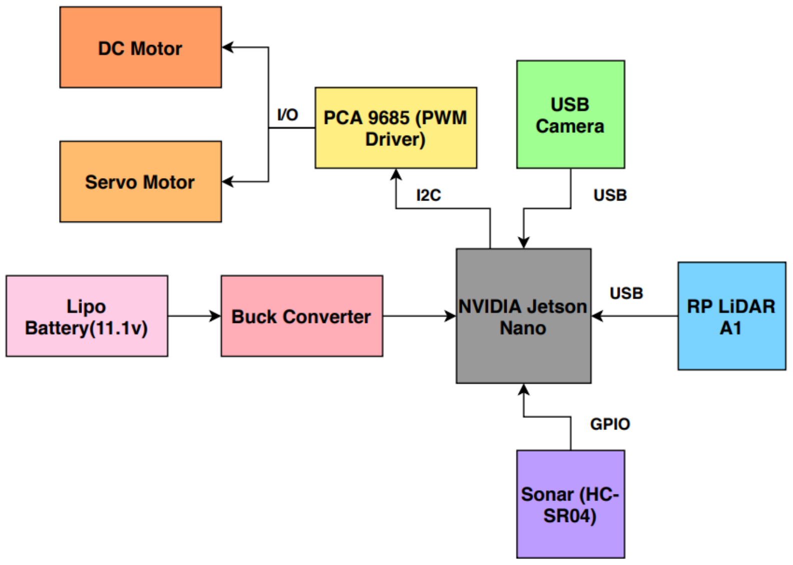


Fig. 4.1. System Hardware Architecture

4.2 Software Architecture

The racecar follows the software architecture as shown in the Fig. 4.2 As Jetson

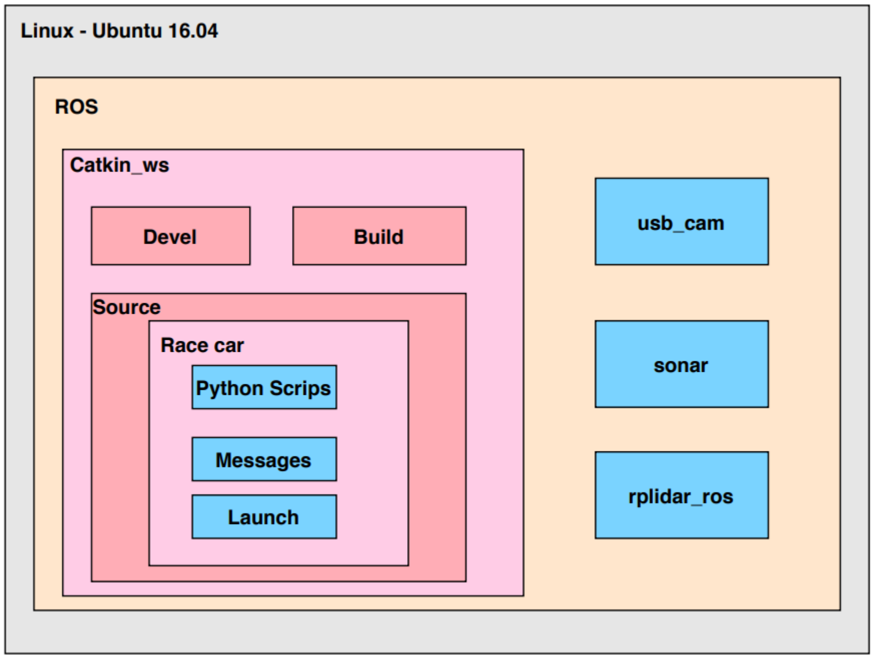


Fig. 4.2. System Software Architecture

Nano is our on-board computer, Ubuntu 16.04 is its default operative system. Robot Operative System (ROS-Melodic) is installed on the Nano. It includes “catkin ws” workspace to develop and build the framework. ROS gives different packages that empower a ready to utilize software, generally depending on parameter configuration. RPLiDAR, USB CAM were the packages that were used in this implementation [1].

Build, Devel and Source are the three directories in catkin workspace. Internally, the source directory contains message folder, launch folder, and all the designed python scripts. Message folder is responsible for saving the message types that were passed through topics. In addition the launch folder permits to invent tailored launch files that initiates multiple nodes, yaml files, config files at the same time [1].

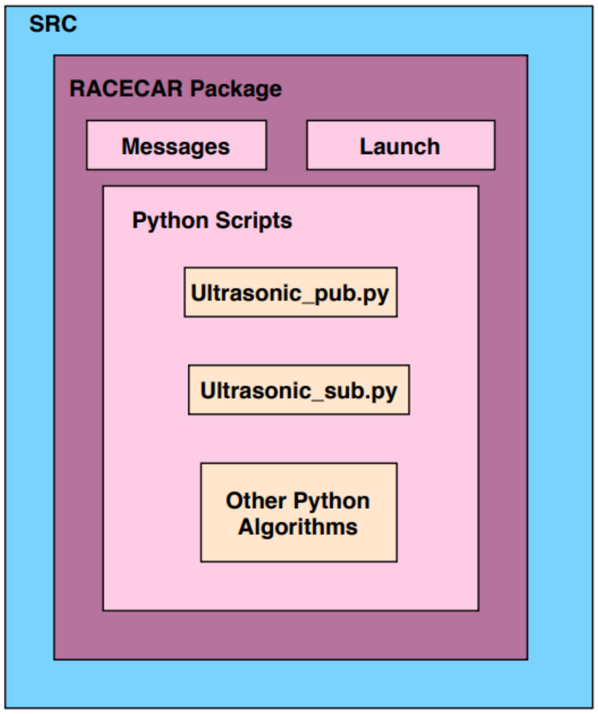


Fig. 4.3. Source Directory of Catkin Workspace

4.3 PCA 9685 (PWM Driver) Interface

A straightforward method to control Servo and DC motor on the Jetson Nano is to associate it by means of I2C communication to a PWM Driver as there is no direct connection between both of them. So, the PWM Driver, takes the steering and throttle values as input commands from the Nano and controls the servo and DC motor of the race car for its steering and throttle respectively [32].

With this driver, an extension can be established between Nano and the electronic speed regulator. Here, the PCA9685 sends Pulse Width Modulation, digital pulses to DC motor and servo, contingent upon the data it received from the Nano.

So the data is sent using I2C protocol from Jetson Nano to PCA9685. The adafruits servokit library has the python commands, where it has the capability of converting the received messages of throttle and steering, into its corresponding

PWM signals. The transformation, depends on throttle values ranging from DASH to DASH, where the 0 represents the stopping value of the car, and the steering values ranging from 55 degrees to 145 degrees, 55 - 99 degrees is turning left, and 101 – 145 degrees is turning right, and 100 degrees is heading straight.

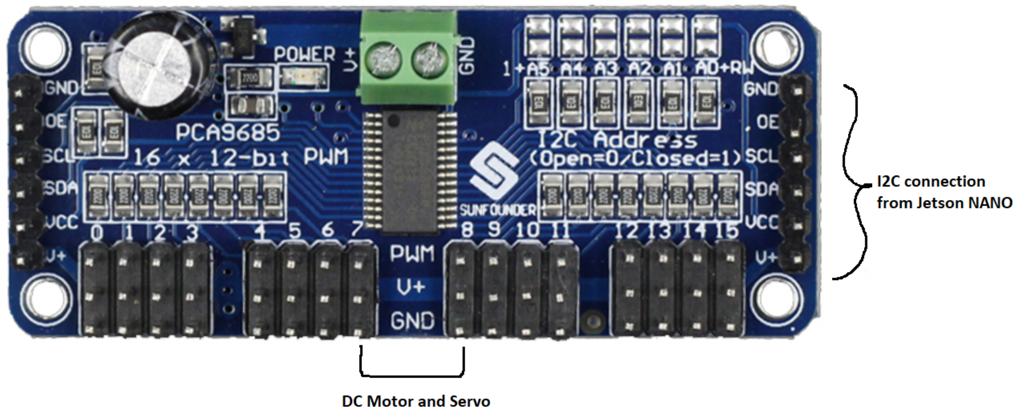


Fig. 4.4. PCA 9685 connection representation with Jetson NANO, DC Motor and Servo

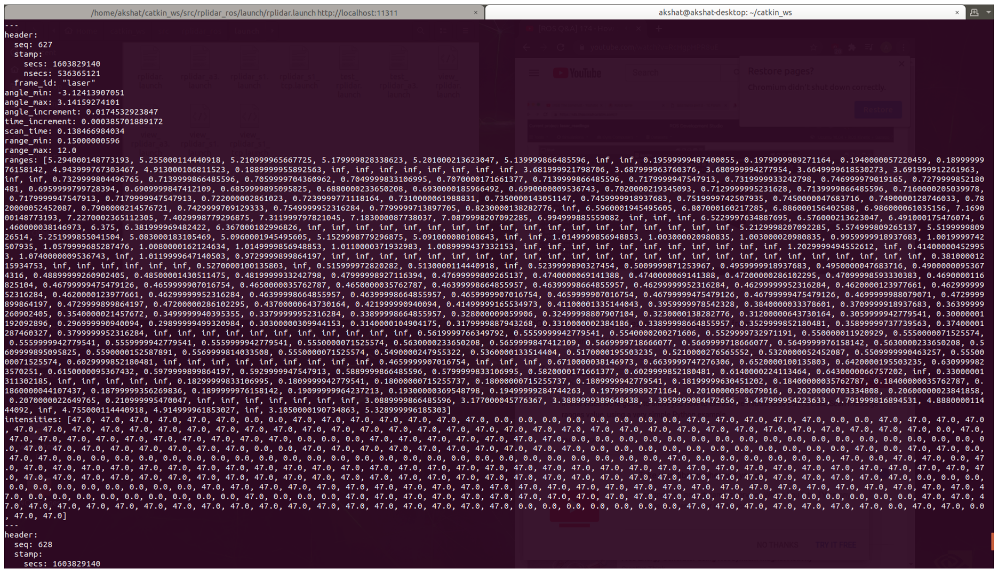
4.4 LiDAR Scanning

The 2D scan data from RP LiDAR sends the data to the Jetson Nano, covering an entire range of 360 degrees of its surroundings. The sample duration of this LiDar is 0.15ms and it is connected via USB to Jetson NANO.

In Fig. 4.5. depicts that Laser Scan topic emits different data values like anglemin, anglemax, angleincrement, ranges and so on.

4.5 Camera Interpretation

The images from USB camera are read using the open CV functions. Depending on the application we play with the pixels and apply lot of image processing techniques on them.

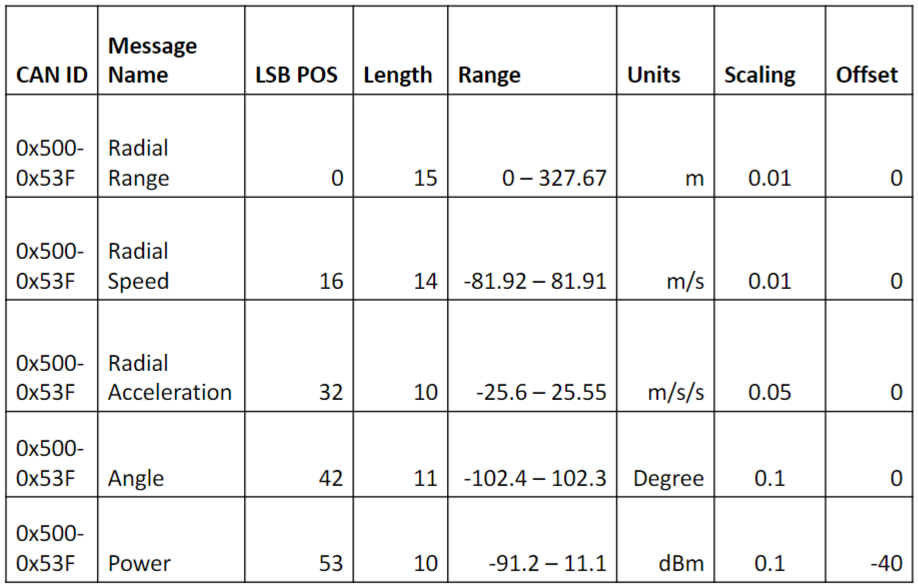

/home/akshat/catkin_ws/src/rplidar_ros/launch/rplidar.launch http://localhost:11311
akshat@akshat-desktop: ~/catkin_ws

```
---
header:
  seq: 627
  stamp:
    secs: 1603829140
    nsecs: 536365121
  frame_id: "laser"
angle_min: -3.12413907051
angle_max: 3.14159274101
angle_increment: 0.0174532923847
time_increment: 0.000385701889172
scan_time: 0.138466984034
range_min: 0.15000000596
range_max: 12.0
ranges: [5.294000148773193, 5.255000114440918, 5.210999965667725, 5.179999828338623, 5.201000213623047, 5.139999866485596, inf, inf, 0.19599999487400055, 0.1979999989271164, 0.1940000057220459, 0.18999999
76158142, 4.943999767303467, 4.913000106811523, 0.1889999955892563, inf, inf, inf, inf, inf, inf, inf, inf, 3.681999921798706, 3.687999963760376, 3.680999994277954, 3.6649999618530273, 3.691999912261963,
inf, inf, 0.7329999804496765, 0.7139999866485596, 0.7059999704360962, 0.7049999833106995, 0.7070000171661377, 0.7139999866485596, 0.7179999947547913, 0.7319999933242798, 0.746999979019165, 0.7279999852180
481, 0.6959999799728394, 0.6909999847412109, 0.6859999895095825, 0.6880000233650208, 0.6930000185966492, 0.6990000009536743, 0.7020000219345093, 0.7129999995231628, 0.7139999866485596, 0.7160000205039978,
 0.7179999947547913, 0.7179999947547913, 0.722000002861023, 0.7239999771118164, 0.7310000061988831, 0.7350000143051147, 0.7459999918937683, 0.7519999742507935, 0.7450000047683716, 0.7490000128746033, 0.78
20000052452087, 0.7900000214576721, 0.7429999709129333, 0.7549999952316284, 0.7799999713897705, 0.8230000138282776, inf, 6.5960001945495605, 6.807000160217285, 6.886000156402588, 6.986000061035156, 7.1690
00148773193, 7.2270002365112305, 7.4029998779296875, 7.311999797821045, 7.183000087738037, 7.0879998207092285, 6.994999885559082, inf, inf, inf, 6.5229997634887695, 6.576000213623047, 6.491000175476074, 6
.460000038146973, 6.375, 6.381999969482422, 6.367000102996826, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, 5.2129998207092285, 5.574999809265137, 5.5199999809
26514, 5.251999855041504, 5.083000183105469, 5.0960001945495605, 5.1529998779296875, 5.091000080108643, inf, inf, 1.0149999856948853, 1.003000020980835, 1.003000020980835, 0.9959999918937683, 1.0019999742
507935, 1.0579999685287476, 1.0080000162124634, 1.0149999856948853, 1.0110000371932983, 1.0089999437332153, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, 1.2029999494552612, inf, 0.41400000452995
3, 1.0740000009536743, inf, 1.0119999647140503, 0.9729999899864197, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, 0.381000012
15934753, inf, inf, inf, inf, inf, inf, 0.5270000100135803, inf, 0.515999972820282, 0.5130000114440918, inf, 0.5239999890327454, 0.5009999871253967, 0.4959999918937683, 0.4950000047683716, 0.4900000095367
4316, 0.48899999260902405, 0.48500001430511475, 0.4819999933242798, 0.47999998927116394, 0.47699999809265137, 0.4740000069141388, 0.4740000069141388, 0.47200000286102295, 0.47099998593330383, 0.4690000116
825104, 0.46799999475479126, 0.4659999907016754, 0.4650000035762787, 0.4650000035762787, 0.46399998664855957, 0.46399998664855957, 0.46299999952316284, 0.46299999952316284, 0.4620000123977661, 0.462999999
52316284, 0.4620000123977661, 0.46299999952316284, 0.46399998664855957, 0.46399998664855957, 0.4659999907016754, 0.4659999907016754, 0.46799999475479126, 0.46799999475479126, 0.4699999988079071, 0.4729999
899864197, 0.4729999899864197, 0.47200000286102295, 0.43700000643730164, 0.421999990940094, 0.41499999165534973, 0.41100001335144043, 0.3959999978542328, 0.3840000033378601, 0.3709999918937683, 0.36399999
260902405, 0.3540000021457672, 0.3499999940395355, 0.33799999952316284, 0.33899998664855957, 0.328000009059906, 0.32499998807907104, 0.3230000138282776, 0.31200000643730164, 0.3059999942779541, 0.30000001
192092896, 0.296999990940094, 0.29899999499320984, 0.30300000309944153, 0.3140000104904175, 0.3179999887943268, 0.3310000002384186, 0.33899998664855957, 0.3529999852180481, 0.35899999737739563, 0.37400001
287460327, 0.3799999952316284, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, 0.5619999766349792, 0.5559999942779541, 0.5540000200271606, 0.5529999732971191, 0.550000011920929, 0.5550000071525574,
 0.5559999942779541, 0.5559999942779541, 0.5559999942779541, 0.5550000071525574, 0.5630000233650208, 0.5659999847412109, 0.5669999718666077, 0.5669999718666077, 0.5649999976158142, 0.5630000233650208, 0.5
609999895095825, 0.5590000152587891, 0.5569999814033508, 0.5550000071525574, 0.5490000247955322, 0.5360000133514404, 0.5170000195503235, 0.5210000276565552, 0.5320000052452087, 0.5509999990463257, 0.55500
00071525574, 0.6029999852180481, inf, inf, inf, inf, inf, inf, inf, inf, inf, 0.4659999907016754, inf, inf, inf, 0.6710000038146973, 0.6639999747276306, 0.6520000100135803, 0.6420000195503235, 0.630999982
3570251, 0.6150000095367432, 0.5979999899864197, 0.5929999947547913, 0.5889999866485596, 0.5799999833106995, 0.5820000171661377, 0.6029999852180481, 0.6140000224113464, 0.6430000066757202, inf, 0.33000001
311302185, inf, inf, inf, inf, inf, 0.18299999833106995, 0.1809999942779541, 0.18000000715255737, 0.18000000715255737, 0.1809999942779541, 0.18199999630451202, 0.18400000035762787, 0.18400000035762787, 0.
1860000044107437, 0.18799999356269836, 0.1899999976158142, 0.19099999964237213, 0.19300000369548798, 0.19499999284744263, 0.1979999989271164, 0.20100000500679016, 0.20200000703334808, 0.20600000023841858,
 0.2070000022649765, 0.210999995470047, inf, inf, inf, inf, inf, inf, inf, 3.0889999866485596, 3.177000045776367, 3.3889999389648438, 3.3959999084472656, 3.447999954223633, 4.791999816894531, 4.8880000114
44092, inf, 4.755000114440918, 4.914999961853027, inf, 3.1050000190734863, 5.328999996185303]
intensities: [47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0
, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 47.0,
 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 0.0, 47.0, 47.0, 0.0,
 47.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 0.0, 47.0, 47.0, 0.0, 47
.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47
.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 4
7.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 0.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 47.0, 4
7.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 47.0, 0.0, 47.0, 47.0, 0.0
, 47.0, 47.0]
---
header:
  seq: 628
  stamp:
    secs: 1603829140
```

Fig. 4.5. /Scan Topic Raw Data

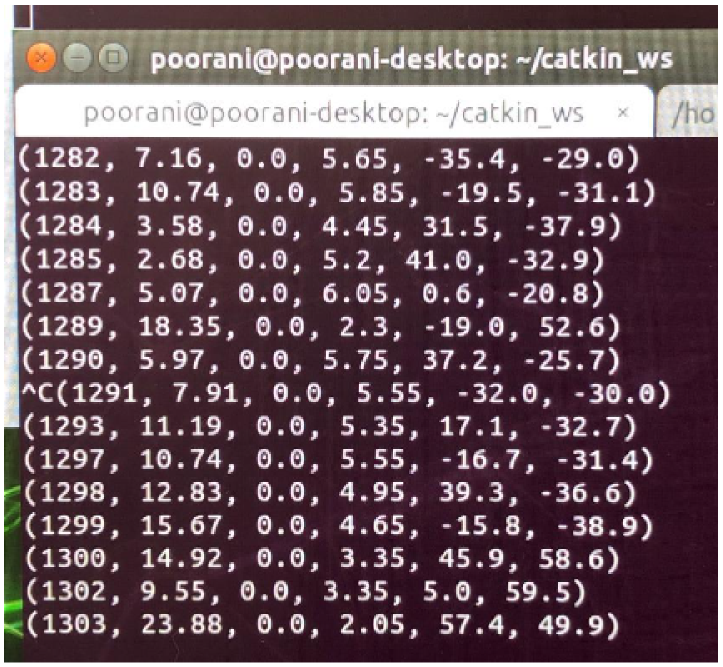
4.6 CAN Decoding From RADAR

The data sent from the radar is in the hexadecimal format and it needs to be decoded into an id based communication. Fig.4.6 represents the CAN ID and its corresponding data values and Fig.4.7 shows the converted can data in ROS environment.



CAN ID	Message Name	LSB POS	Length	Range	Units	Scaling	Offset
0x500-0x53F	Radial Range	0	15	0 – 327.67	m	0.01	0
0x500-0x53F	Radial Speed	16	14	-81.92 – 81.91	m/s	0.01	0
0x500-0x53F	Radial Acceleration	32	10	-25.6 – 25.55	m/s/s	0.05	0
0x500-0x53F	Angle	42	11	-102.4 – 102.3	Degree	0.1	0
0x500-0x53F	Power	53	10	-91.2 – 11.1	dBm	0.1	-40

Fig. 4.6. Data Ranges and their Corresponding CAN ID's


poorani@poorani-desktop: ~/catkin_ws
poorani@poorani-desktop: ~/catkin_ws
/ho

```
(1282, 7.16, 0.0, 5.65, -35.4, -29.0)
(1283, 10.74, 0.0, 5.85, -19.5, -31.1)
(1284, 3.58, 0.0, 4.45, 31.5, -37.9)
(1285, 2.68, 0.0, 5.2, 41.0, -32.9)
(1287, 5.07, 0.0, 6.05, 0.6, -20.8)
(1289, 18.35, 0.0, 2.3, -19.0, 52.6)
(1290, 5.97, 0.0, 5.75, 37.2, -25.7)
^C(1291, 7.91, 0.0, 5.55, -32.0, -30.0)
(1293, 11.19, 0.0, 5.35, 17.1, -32.7)
(1297, 10.74, 0.0, 5.55, -16.7, -31.4)
(1298, 12.83, 0.0, 4.95, 39.3, -36.6)
(1299, 15.67, 0.0, 4.65, -15.8, -38.9)
(1300, 14.92, 0.0, 3.35, 45.9, 58.6)
(1302, 9.55, 0.0, 3.35, 5.0, 59.5)
(1303, 23.88, 0.0, 2.05, 57.4, 49.9)
```

Fig. 4.7. Decoded CAN Values from Hexadecimal to Decimal

5. IMPLEMENTATION

5.1 Modeling the RACECAR

An autonomous vehicle need to understand its instantaneous state in order to make decisions on how to control the vehicle. In any robotic framework, it is always necessary to describe the evolution of the state of the system over time. The structure of the model is represented as follows:

$$\dot{x} = f(x, u)$$

Here, x describe the state of the system and u describes control inputs. The control inputs for the Racecar are steering angle and throttle setpoints, therefore, $u \epsilon R^2$.

5.2 Notations used While Describing the Model

Basic notations utilized in depicting the vehicle dynamics:

- δ: steering angle set-point. Typically, the units are in radians.
- ψ: throttle values. Typically the units are in m/s, explicitly the speed at which the back tires would be moving comparative with the body frame at the contact point between the tire and the wheel. From this definition, it is interpreted that a throttle set-point of 1, then under no slip conditions we can say that the race car is having a tangential velocity of 1 m/s [33].

Fig. 5.1. shows the ideal race car representation (on the left), and its corresponding bicycle model(on the right). Here, l_a and l_b are the distances from the car center of mass to the front and back tires, respectively.

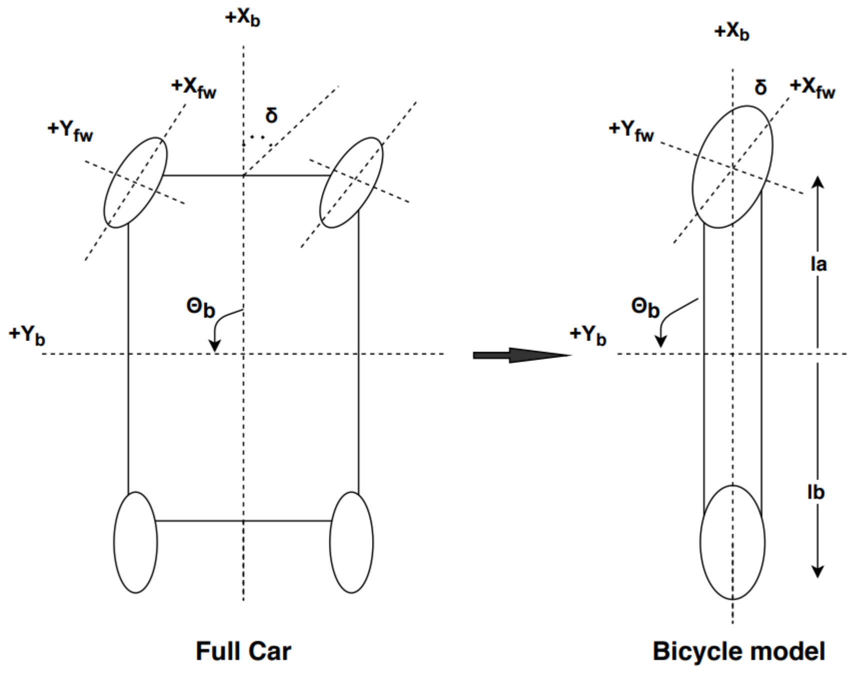


Fig. 5.1. Bicycle Model [33]

5.2.1 Coordinate Frames

World Frame

It is the top level frame in our modeling. Here, this model is considered to be stationary w.r.t the known system of a map. The subscript 'w' is used to represent the world frame [33].

Body Frame

This is fixed frame w.r.t the race-car. The origin of this coordinate frame is either at the race car's center of mass or at base link pose which is found straight forwardly between the back tires. The forward direction of the car is considered as positive x axis, and the leftward direction from the car is considered as positive y axis. The subscript ‘b’ is used to represent the body frame [33].

Wheel Frames

As we considered bicycle model, the frames of front and rare tires are only taken into consideration. The back tire frame is like the body frame, except for translation. The front tire’s frame is likewise made an interpretation w.r.t the body frame, however it is additionally rotated by precisely the steering angle ψ. The subscripts ‘fw’ and ‘rw’ represents the front wheel and back wheel frames respectively [33].

5.3 Control Methods

Different algorithms can be designed to control my race car. The steering stability on the race car was achieved by using a Proportional Integral, PI, controller.

5.3.1 PI Control

The most often method for controlling a car is the proportional integral, or a PI controller in which it gives the steering command corresponds to its error term from the target [34]. The Fig. 5.2. shows the block diagram of a PI controller. The mathematical representation of the control error is represented as follows:

$$controloutput(t) = K_p * e(t) + K_I * \int e(t)dt$$

Here, the output steering angle is the result of P term added to a integral of I term. The K_p and K_i are experimentally derived constants. Here, K_p is directly multiplied with the amount of error and its value is usually varied from $0 < K_p < 10$. Similarly, K_i is multiplied to the integral of the error i.e, the sum of all the instantaneous error values at that point. Even the K_i values are varied from $0 < K_i < 1$ [34].

These K_p and K_i values are tuned manually by a trail and error basis with respect to the driving scenario race car.

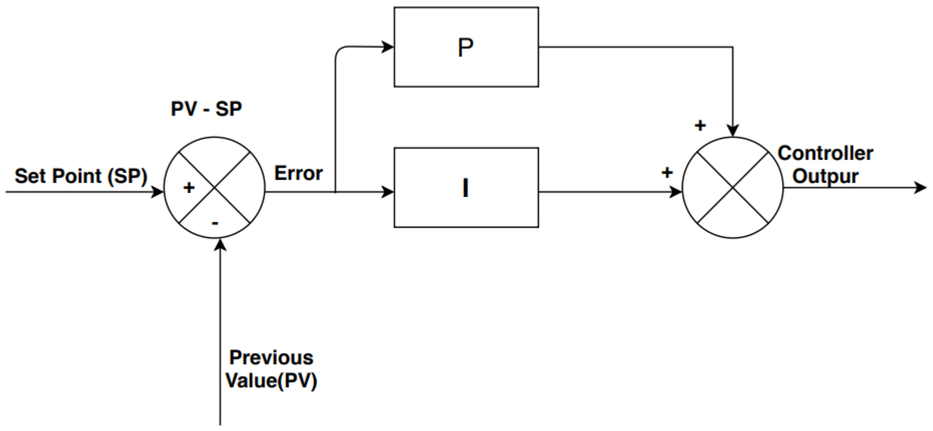


Fig. 5.2. PI Controller [34]

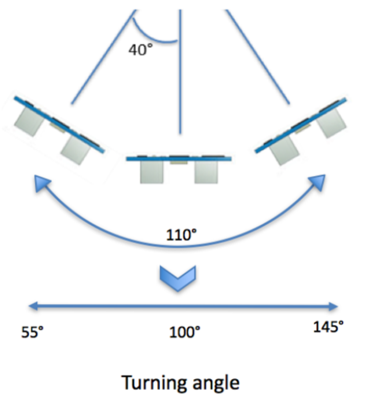
5.4 Obstacle avoiding RACECAR using Jetson Nano and Ultrasonic sensors

On the RACECAR, we have used three ultrasound distance sensors, HC-SRO4s. The placement of these sensors are shown in the Fig. 5.3. These are placed in such a way to attain the best coverage area along the forward direction. It was chosen by means of experimental strategies that these positions shown in the figure were ideal for obstacle recognition when three ultrasonic sensors were being used.

Hardware connections were made by connecting the Jetson Nano GPIO pins to the Trigger, Echo, VCC and GND pins of HC-SRO4s respectively.

It was assumed that all the necessary packages, message types and so on were already imported into the ROS environment. The algorithm described in Fig. 5.4. and Fig. 5.5. is a simple method for collision avoidance using these three Ultrasonic sensors. The python scripts are written ROS publisher and subscriber format. In the sonar publisher algorithm, we are calculating the distances from center, right and left sonars and publishing those distance values to the chatter topic.

In the sonar subscriber algorithm, first we subscribe to the chatter topic and get the calculated distance values from all the three (left, center and right) sensors. In



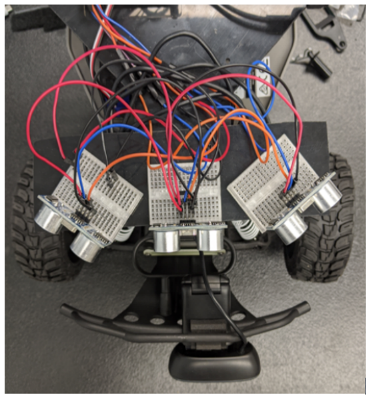


Fig. 5.3. Position of Ultrasonic Sensors

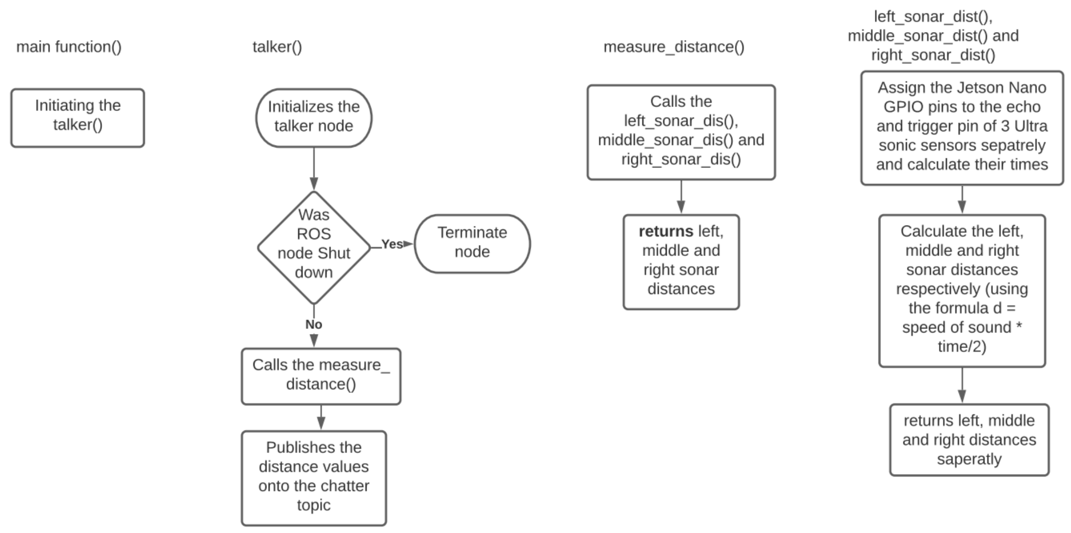

main function()
Initiating the talker()
talker()
Initializes the talker node
Was ROS node Shut down
Yes
Terminate node
No
Calls the measure_ distance()
Publishes the distance values onto the chatter topic
measure_distance()
Calls the left_sonar_dis(), middle_sonar_dis() and right_sonar_dis()
returns left, middle and right sonar distances
left_sonar_dist(), middle_sonar_dist() and right_sonar_dist()
Assign the Jetson Nano GPIO pins to the echo and trigger pin of 3 Ultra sonic sensors sepatrely and calculate their times
Calculate the left, middle and right sonar distances respectively (using the formula d = speed of sound * time/2)
returns left, middle and right distances saperatly

Fig. 5.4. Sonar Publisher Algorithm

the next step, we considered a threshold value of 50cm and all the distances are compared with this value. Finally, the flow of comparing threshold value and taking the necessary action is explained in the Fig. 5.5.

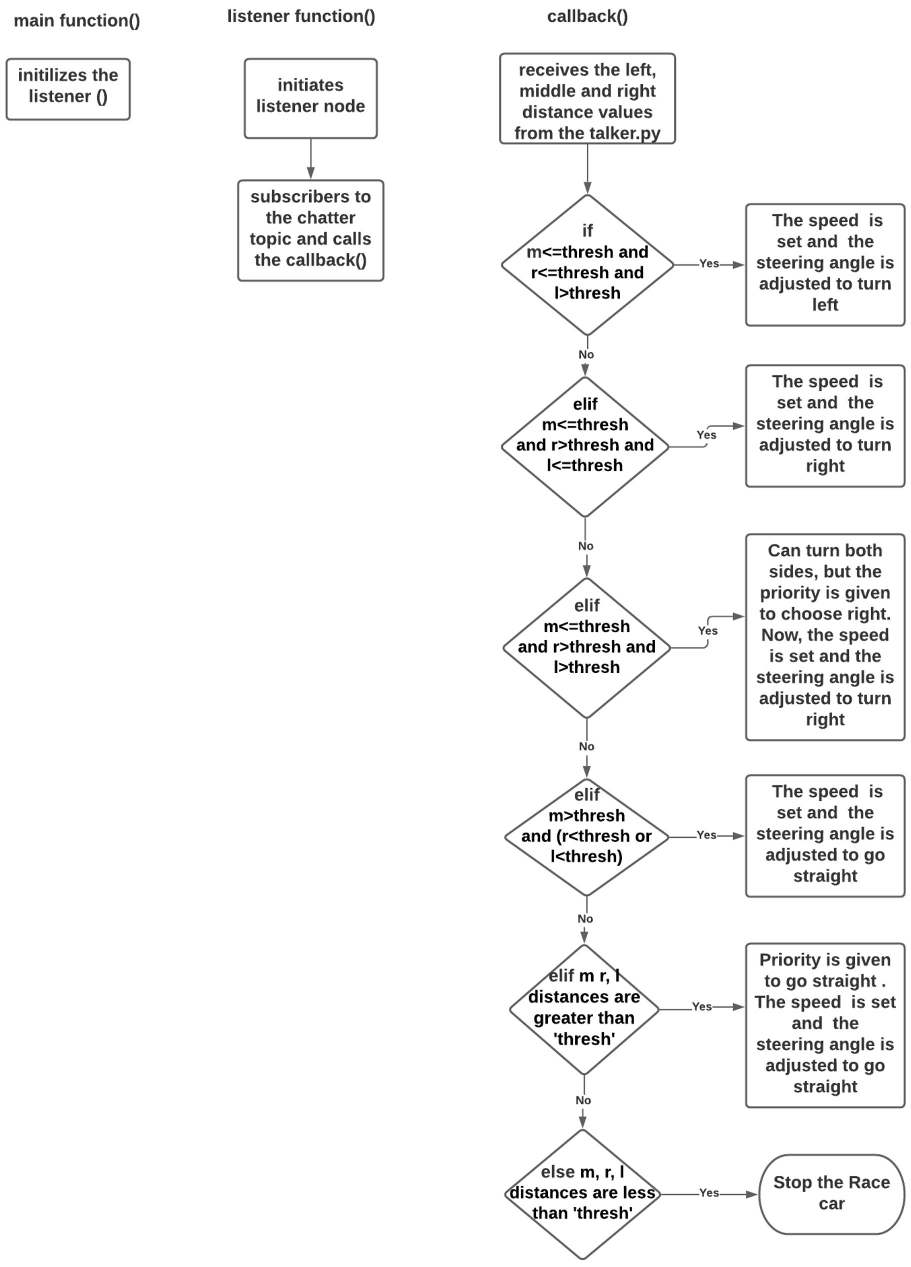

main function()
initilizes the listener ()
listener function()
initiates listener node
subscribers to the chatter topic and calls the callback()
callback()
receives the left, middle and right distance values from the talker.py
if m<=thresh and r<=thresh and l>thresh
Yes
The speed is set and the steering angle is adjusted to turn left
No
elif m<=thresh and r>thresh and l<=thresh
Yes
The speed is set and the steering angle is adjusted to turn right
No
elif m<=thresh and r>thresh and l>thresh
Yes
Can turn both sides, but the priority is given to choose right. Now, the speed is set and the steering angle is adjusted to turn right
No
elif m>thresh and (r<thresh or l<thresh)
Yes
The speed is set and the steering angle is adjusted to go straight
No
elif m r, l distances are greater than 'thresh'
Yes
Priority is given to go straight . The speed is set and the steering angle is adjusted to go straight
No
else m, r, l distances are less than 'thresh'
Yes
Stop the Race car

Fig. 5.5. Sonar Subscriber Algorithm

5.5 Lane following RACECAR using Jetson Nano and USB Camera

On the RACECAR, we have used Logitech C270 USB camera for lane detection.

The algorithm for lane detection was completely based on computer vision and image processing techniques [35]. We used OpenCV library with python interface on ROS platform for developing this algorithm. It is assumed that all the necessary packages, messages were imported into the ROS environment.

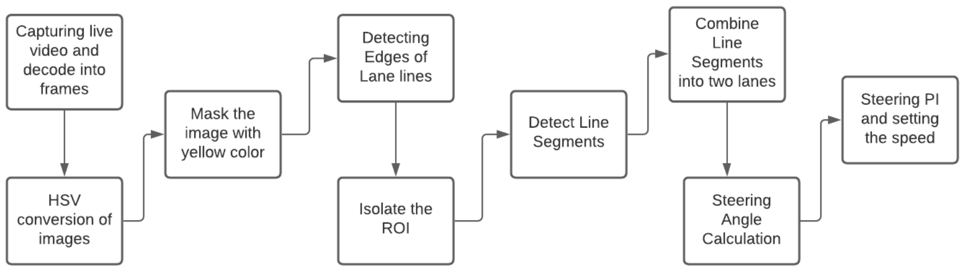

Capturing live video and decode into frames
HSV conversion of images
Mask the image with yellow color
Detecting Edges of Lane lines
Isolate the ROI
Detect Line Segments
Combine Line Segments into two lanes
Steering Angle Calculation
Steering PI and setting the speed

Fig. 5.6. Lane Detection Block Diagram

5.5.1 Capturing and Decoding the Live Video

Live video is captured using VideoCapture() object in open CV and then it is decoded into frames.

5.5.2 HSV Conversion of an Image

In this step the captured video frames, which are in BGR color space are converted into HSV color space. The main reason behind converting into an HSV format is that, in BGR format various parts of the yellow tape might be illuminated with various light conditions, resulting them to appear as darker yellow or lighter yellow. However, in HSV color space, the Hue segment will deliver the entire yellow tape into one color irrespective of its shading.

5.5.3 Mask the Image with Yellow Color

Now, from the HSV image, the upper and lower boundary ranges for yellow color are specified, so that only yellow color is lifted in the image. The output from this step is a masked black and white image with yellow part lifted as white space.

5.5.4 Detecting Edges of Lane Lines

Canny Edge Detector is used to find the edges in the masked image. It computes gradient in all directions of the masked image and traces the edges with large change in the intensity. OpenCV library directly has canny edge detector function that can be used directly.

5.5.5 Isolate Region of Interest

While detecting lanes, we only concentrate on the screen that is near to our lanes and rest of the screen can be ignored. So, the top of the screen is cropped out resulting in the region only with the lane lines.

5.5.6 Detect Line Segments

In this step, line segments are detected from the ROI image. We used 'Hough Transform' function to detect the line segments which is directly available from the OpenCV library. The output from this step is the end point coordinates (x1, y1) followed by (x2, y2) of each detected line segment. Here, four line segments are detected.

5.5.7 Combining the Detected Four Line Segments into Two Lane Lines

In this step, we combined all the detected line segments with their end point coordinates (x1, y1) and (x2, y2) are combined into two lines. First, we classified the lane segments by their slope values. It was assumed that all the left lane lines should be upward sloping. Similarly, all the right lane lines should be downward sloping. Next, from the classified groups of left and right lane segments, average of the slopes and the intercepts are considered to finally classify them into left and right lanes.

5.5.8 Steering Angle Calculation

From the above step we have coordinates of the detected left and right lanes, so the next step is to steer the racecar to stay with in the lane lines. By averaging the far end points of both lines, heading direction and steering angle are calculated. Since, the USB Camera is installed in the middle of the racecar, which is pointing straight ahead, the lower end of the heading line is always considered to be on the center of the bottom of the screen.

5.5.9 Steering PI and Setting the Speed

Steering PI Controller is used to tune the values with respect to the calculated steering angle (from the above step) and to stabilize the racecar from wobbling. The speed of the racecar is fixed to 0.5 m/s.

5.6 Track following RACECAR using Jetson Nano and RP LiDAR

The LiDAR has 360 degree scan capability, it scans for the angles from 0 degrees to 360 degrees in anti-clock wise direction, with 0 degrees being the front of the car, as shown in the Fig. 5.7.

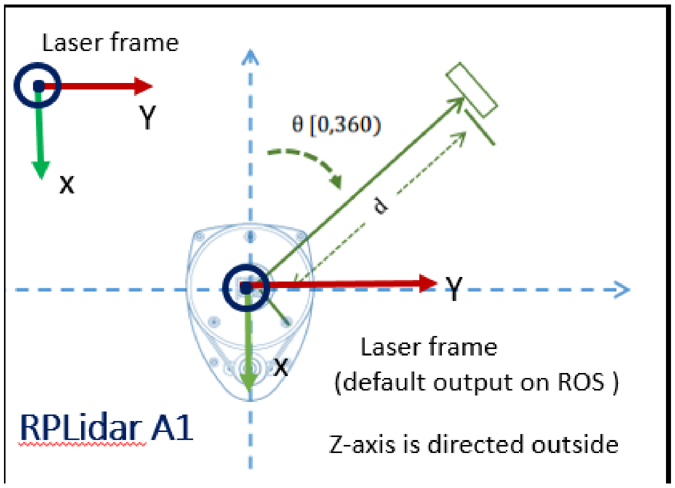


Fig. 5.7. RPLiDAR A1 [30]

RACECAR is mounted with RP LiDAR A1 to follow the race track. Before even developing an algorithm it is important to come up with a deciding window for our lidar. As shown in the Fig. 5.8. the key parameters to be considered while deciding the window are as follows:

- The Racecar is 27cm wide so, to be on a safer side the window size is considered as 40cm.
- The distance between any object and the race car should be at least 50cm. That is no object should be in this window whose distance is less than 50cm. Therefore, the blind spot region is considered to be 50cm.
- Using trigonometry θ was found to be 40 degrees.

The RP LiDAR A1 emits the LasarScan topic, which publishes different values like angle-min, angle-max, angle-increment, ranges and so on as shown in Fig. 5.9 in its ROS environment. So, by subscribing to this topic, it is possible to determine the distance at a particular angle.

This algorithm depicts a simple technique for the RACECAR to follow the race track. The program is written in python with ROS subscriber format. As sensor msgs has LaserScan topic, there is no need of an publisher program.

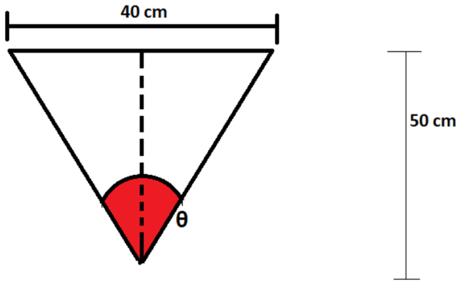
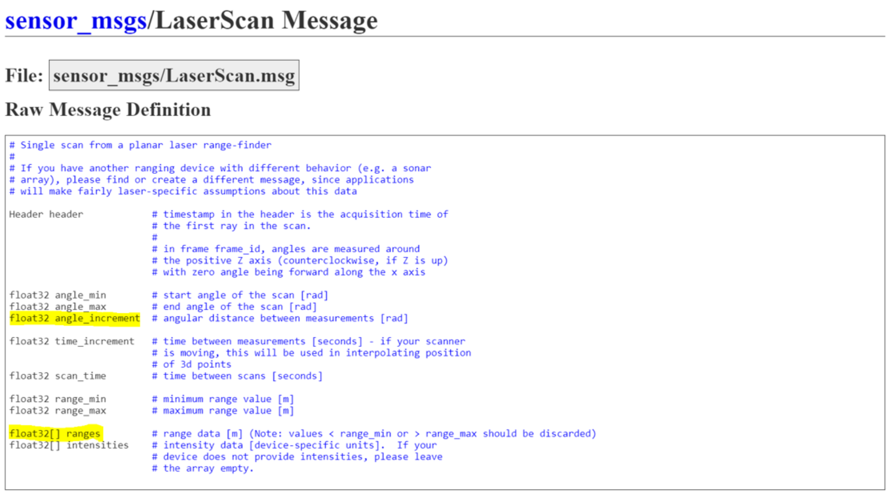


Fig. 5.8. Deciding Window for LiDAR



sensor_msgs/LaserScan Message

File: sensor_msgs/LaserScan.msg

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Fig. 5.9. /LaserScan Topic

It assumed that the car moves only in the forward direction only. So, in the first step the scan angles are sliced to the front part only, that is [0 to 90] and [270 to 360] and then they are reversed and merged from [90 to 0] to (360 to 270]. The second step would be defining a threshold for range values and the maximum gap value (deciding the window). The values that worked best for me are **0.5 meters** for minimum threshold, and **40 centimeters** as the maximum gap value. In the third step all consecutive range values are appended to a empty array and then sorted them in the ascending order. From this values the list of largest gap array was taken. For-

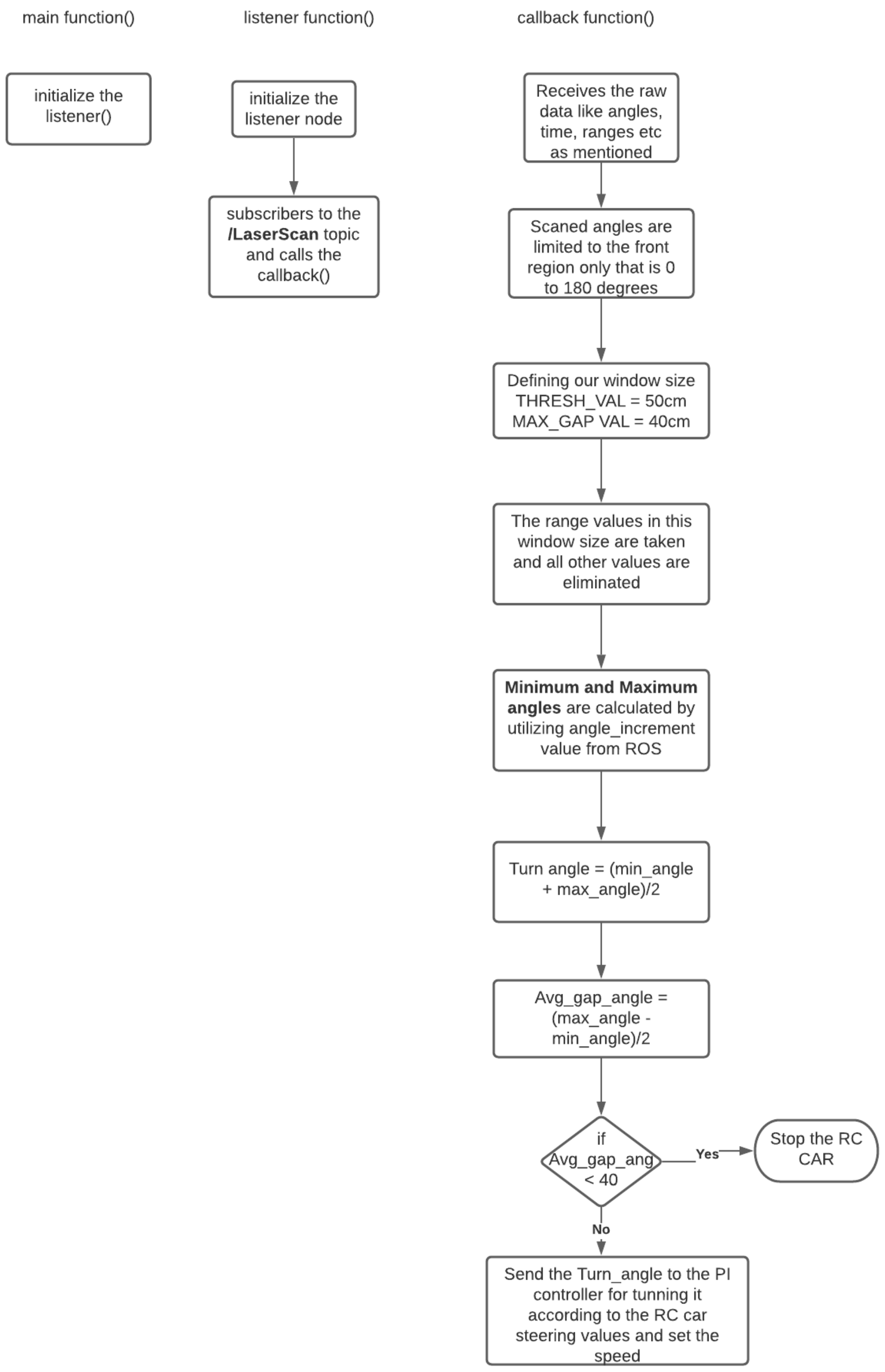

main function()
listener function()
callback function()
initialize the listener()
initialize the listener node
subscribers to the /LaserScan topic and calls the callback()
Receives the raw data like angles, time, ranges etc as mentioned
Scaned angles are limited to the front region only that is 0 to 180 degrees
Defining our window size THRESH_VAL = 50cm MAX_GAP VAL = 40cm
The range values in this window size are taken and all other values are eliminated
Minimum and Maximum angles are calculated by utilizing angle_increment value from ROS
Turn angle = (min_angle + max_angle)/2
Avg_gap_angle = (max_angle - min_angle)/2
if Avg_gap_ang < 40
Yes
Stop the RC CAR
No
Send the Turn_angle to the PI controller for tunning it according to the RC car steering values and set the speed

Fig. 5.10. LiDAR Subscriber Algorithm

tunately, /LaserScan topic from the LiDAR in the ROS gives us the angle increment value for **each sec**. Hence, the maximum and the minimum angles are calculated. In the fourth step, average gap angle and the turn angle are calculated. In the final step the conditions to move the RACE CAR are given, i.e, if average gap is less than maximum angle then STOP the car. Else the steering is tuned according to the 'turn angle' by integrating proportional integral logic (i.e steering PI control) with it. The formulas used in this algorithm are:

minimum angle = largest gap[0] * angle increment

maximum angle = largest gap[-1] * angle increment

avg gap angle = (maximum angle - minimum angle)/2

turn angle = minimum angle/2 + maximum angle/2

6. RESULTS

This section shows the outcomes accomplished all through the development of the RACECAR using different sensors.

First of all the racecar is successfully equipped with Ultrasonic sensors, LiDAR and Camera and an on-board computer. All the above mentioned sensors along with motor and servo control are successfully interfaced with ROS (Robot Operating System).

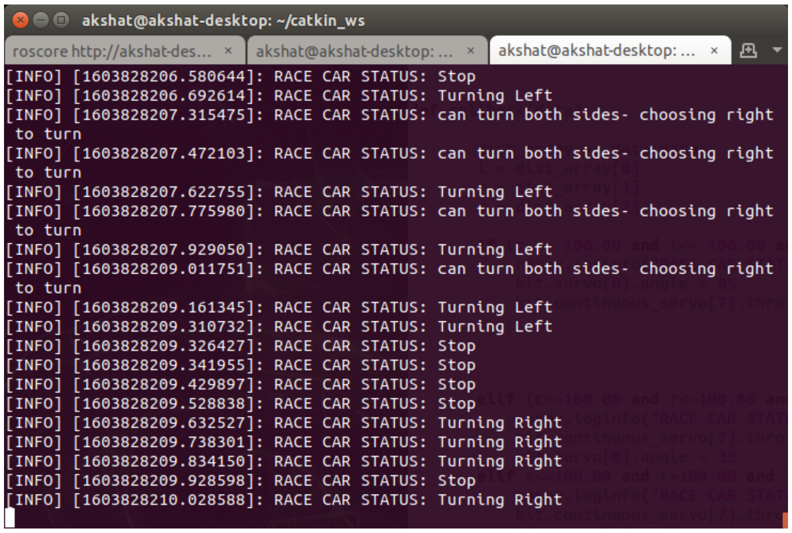
6.1 Obstacle avoiding RACECAR using Jetson Nano and Ultrasonic sensors

Racecar was able to avoid the obstacles and travels with a speed of 0.5m/s. It does the actions according to the defined algorithm.

The following exhibit the outcomes acquired using the algorithm developed for Ultrasonic sensor.

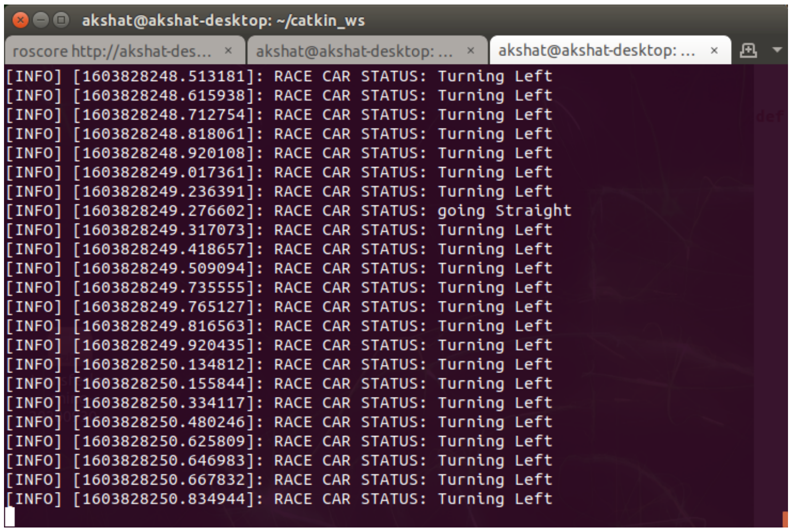
- RACECAR is turning **Left** if the distances from the middle and right sensors are less than 100cm but the distance from the left sensor is greater than 100cm.
- RACECAR is turning **Right** if the distances from the middle and left sensors are less than 100cm but the distance from the right sonar is greater than 100cm.
- RACECAR is turning **Right** if the distances from the middle less than 100cm but the distance from the left and right sonars is greater than 100cm. Priority is given to choose right.
- RACECAR is going **Straight** if the distances from middle and right sonars are less than 100cm but the distance from the middle sensor is greater than 100cm.

- RACECAR is going **Straight** if the distances from all the three sensors are greater than 100cm.
- RACECAR **Stops** if the distances from all the three sensors is than 100cm.



```
[INFO] [1603828206.580644]: RACE CAR STATUS: Stop
[INFO] [1603828206.692614]: RACE CAR STATUS: Turning Left
[INFO] [1603828207.315475]: RACE CAR STATUS: can turn both sides- choosing right
 to turn
[INFO] [1603828207.472103]: RACE CAR STATUS: can turn both sides- choosing right
 to turn
[INFO] [1603828207.622755]: RACE CAR STATUS: Turning Left
[INFO] [1603828207.775980]: RACE CAR STATUS: can turn both sides- choosing right
 to turn
[INFO] [1603828207.929050]: RACE CAR STATUS: Turning Left
[INFO] [1603828209.011751]: RACE CAR STATUS: can turn both sides- choosing right
 to turn
[INFO] [1603828209.161345]: RACE CAR STATUS: Turning Left
[INFO] [1603828209.310732]: RACE CAR STATUS: Turning Left
[INFO] [1603828209.326427]: RACE CAR STATUS: Stop
[INFO] [1603828209.341955]: RACE CAR STATUS: Stop
[INFO] [1603828209.429897]: RACE CAR STATUS: Stop
[INFO] [1603828209.528838]: RACE CAR STATUS: Stop
[INFO] [1603828209.632527]: RACE CAR STATUS: Turning Right
[INFO] [1603828209.738301]: RACE CAR STATUS: Turning Right
[INFO] [1603828209.834150]: RACE CAR STATUS: Turning Right
[INFO] [1603828209.928598]: RACE CAR STATUS: Stop
[INFO] [1603828210.028588]: RACE CAR STATUS: Turning Right
```

Fig. 6.1. Snippet 1 of Result Window using Ultrasonic Sensors



```
[INFO] [1603828248.513181]: RACE CAR STATUS: Turning Left
[INFO] [1603828248.615938]: RACE CAR STATUS: Turning Left
[INFO] [1603828248.712754]: RACE CAR STATUS: Turning Left
[INFO] [1603828248.818061]: RACE CAR STATUS: Turning Left
[INFO] [1603828248.920108]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.017361]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.236391]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.276602]: RACE CAR STATUS: going Straight
[INFO] [1603828249.317073]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.418657]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.509094]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.735555]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.765127]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.816563]: RACE CAR STATUS: Turning Left
[INFO] [1603828249.920435]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.134812]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.155844]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.334117]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.480246]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.625809]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.646983]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.667832]: RACE CAR STATUS: Turning Left
[INFO] [1603828250.834944]: RACE CAR STATUS: Turning Left
```

Fig. 6.2. Snippet 2 of Result Window Using Ultrasonic Sensors

6.2 Lane Following RACECAR using Jetson Nano and USB Camera

Racecar was able to detect and follow the lane and travels with a speed of 0.5m/s. This algorithm uses a robust technique, to control the race car's steering, by utilizing a PI controller. Constantly checking with trial and error approach, the consistent estimations of the PI values are:

- K_i : 0.1
- K_p : 1

The following exhibit the outcomes acquired from the algorithm developed for using USB Camera.

6.2.1 Capturing and Decoding the Live Video

We have created a yellow track at my work place for running the RACECAR uisng the lane following algorithm. Fig. 6.3 shows a decoded RGB frame from the live video.

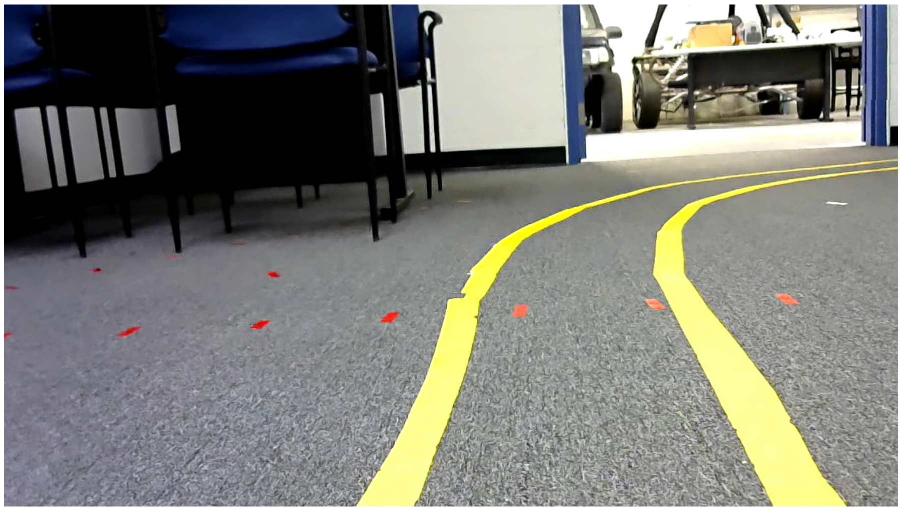


Fig. 6.3. RGB Yellow Lane Image

6.2.2 HSV Conversion of an Image

Fig. 6.3 shows the HSV converted image.

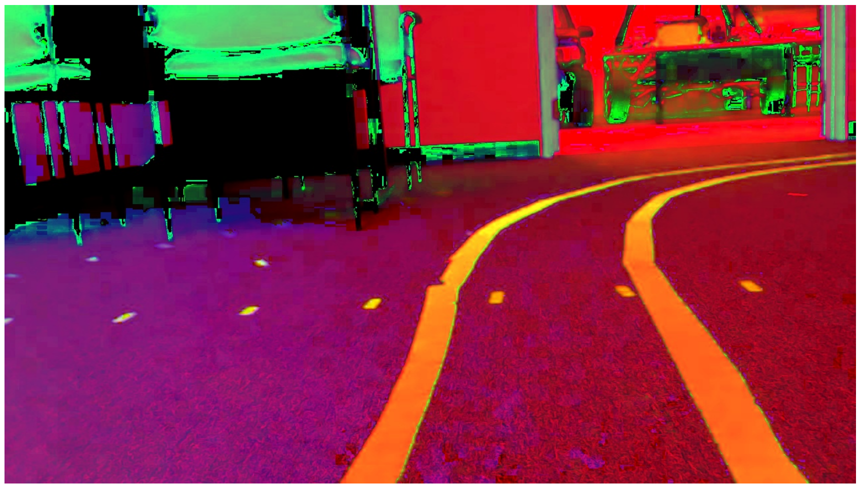


Fig. 6.4. HSV Image

6.2.3 Mask the Image with Yellow Color

Fig. 6.4 shows the masked image with yellow color.

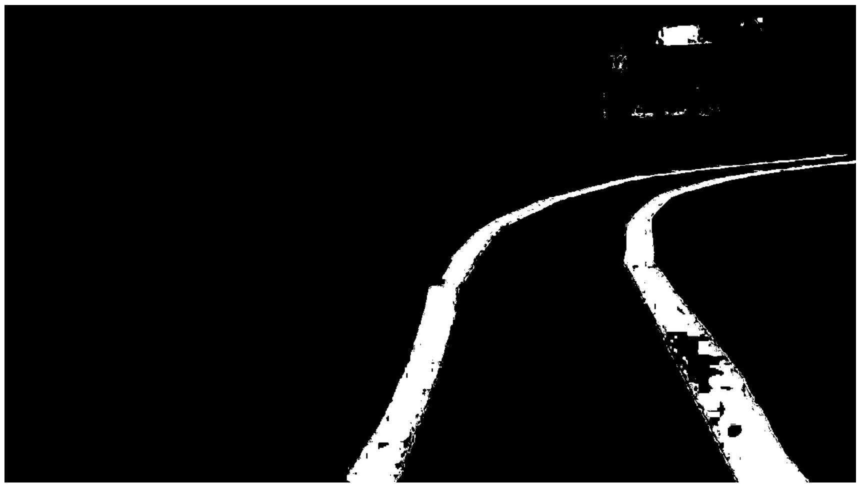


Fig. 6.5. Masked Image with Yellow Color

6.2.4 Detecting Edges of Lane Lines

Fig. 6.5 shows the detected edges from a Canny Edge Detector.

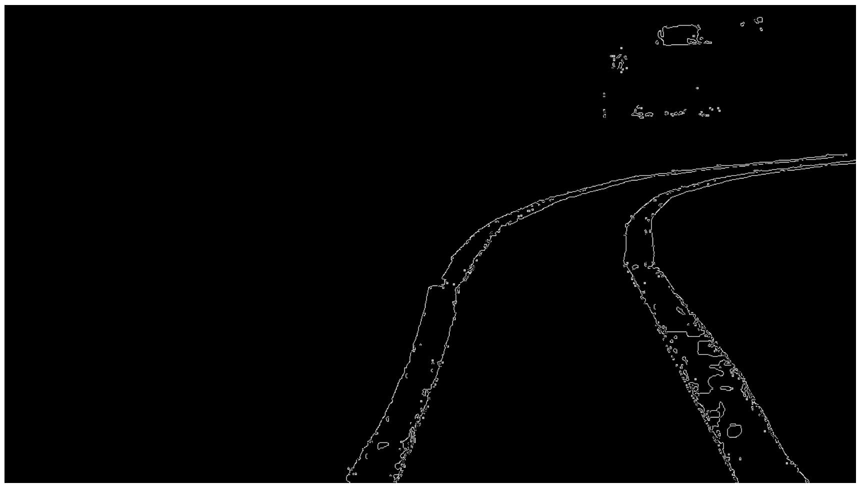


Fig. 6.6. Edge Detected Image

6.2.5 Isolate Region of Interest (ROI)

Fig. 6.6 shows isolated image just with the lanes on the bottom half of the screen and neglecting the rest.

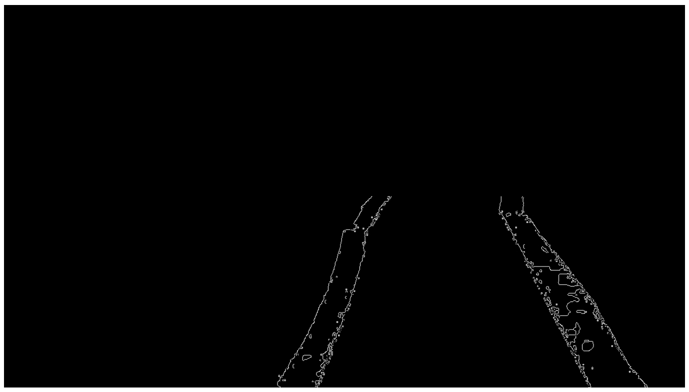


Fig. 6.7. ROI Image

6.2.6 Detect Line Segments

Fig. 6.7 shows the detected line segment using Hough Transform method.

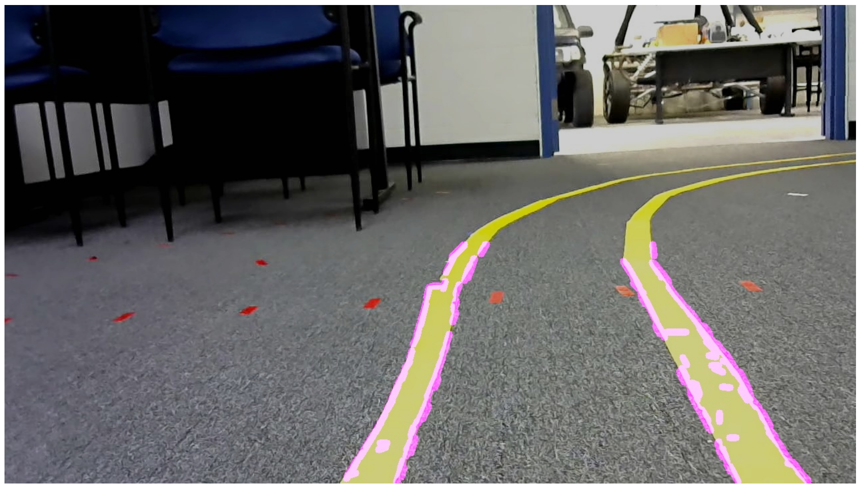


Fig. 6.8. Detected Line Segments Image

6.2.7 Combining the Detected Four Line Segments into Two Lane Lines

Fig. 6.8 shows the image just with the two lanes.

6.2.8 RACECAR Heading Direction

Fig. 6.8 shows the image just with the two lanes.

6.3 Track Following RACECAR using Jetson Nano and RP LiDAR A1

Racecar was able to follow the track and travels with a speed of 0.5m/s. This algorithm uses a robust technique, to control steering of the racecar using a PI controller. By continuous trial and error, the best consistent estimations of the PI values were:

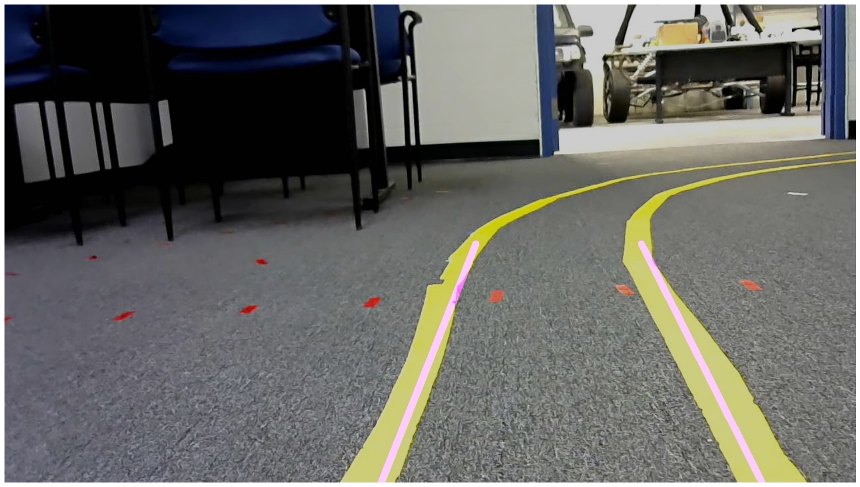


Fig. 6.9. Lane Lines Image

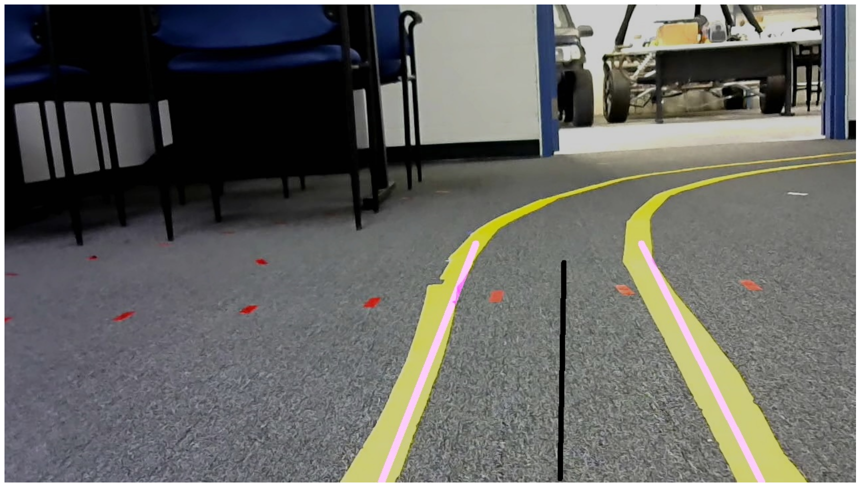


Fig. 6.10. Image with Heading Direction

- K_p : 1
- K_i : 0.1

7. FUTURE WORK

7.1 Speed PI controller

Right now, we are running the RACE CAR at a constant speed of 0.5 m/s. In future, we should definitely increase the speed of the RACE CAR. We need to develop a speed PI controller in order to effectively control the speed at different distances.

7.2 RADAR

RADAR (Radio Detection And Ranging) is a very important sensor in autonomous technology which works exceptionally in bad weather conditions. We can use RADAR in lot of applications like distance and velocity measurement of the front vehicle, obstacle avoidance, adaptive cruise control, blind spot detection, rear cross traffic alert, car door opening warning and so on. In the previous chapters, we basically decoded and analyzed the CAN data from the sensor. The next step would be to implement it on the RACECAR for any of the mentioned applications.

7.3 Sensor Fusion

Multiple sensors may be combined or "Fused" to supply an improved signal from the first single sources. The development can mean better accuracy, availability and the usability of sensed date. Sensor fusion plays and important role within the autonomous system capability and performance.

Sensor fusion uses the sensor information and combines the data streams to the Perception layer. It provides the perception layer with data that can be simpler to formulate the situational awareness algorithms. By using multiple sensors of the same type signal accuracy and data quality can be improved.

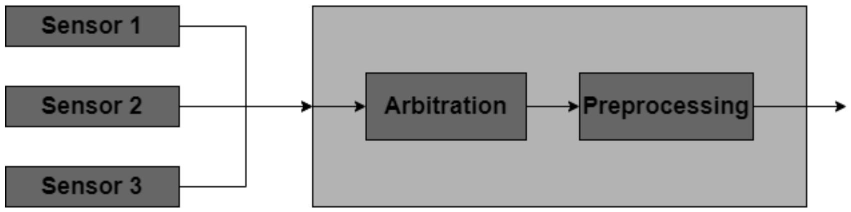


Fig. 7.1. Sensor Fusion

Therefore, the next implementation is to develop a RACECAR which uses the sensor fusion technology. For example, ‘Lane Change Assist’ system using camera and lidar.

REFERENCES

REFERENCES

[1] D. Almeida, "Implementing and tuning an autonomous racing car testbed," Master's thesis, 2019.

[2] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, T. Stahl, L. Hermansdorfer, and M. Lienkamp, "A software architecture for an autonomous racecar," *ResearchGate*, 2019.

[3] Sae international releases updated visual chart for self driving vehicles. [Online; accessed 22-October-2020]. [Online]. Available: https://www.sae.org

[4] S. Liu, L. Li, J. T. S. Wu, and J.-L. Gaudiot, *Creating Autonomous Vehicle Systems*. [S.l.]: MORGAN & CLAYPOOL PUBLISH, 2020.

[5] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," *Integration the VLSI*, 2017.

[6] Kangalow. Racecar/j chassis preparation. [Online; accessed 24-August-2020]. [Online]. Available: https://www.jetsonhacks.com

[7] Kanglow. Racecar/j platform preparation. [Online; accessed 24-August-2020]. [Online]. Available: http://www.jetsonhacks.com

[8] Jetson nano. [Online; accessed 24-August-2020]. [Online]. Available: https://www.nvidia.com

[9] 16-channel 12-bit pwm/servo driver - pca9685. [Online; accessed 24-August-2020]. [Online]. Available: https://www.adafruit.com/product/815

[10] Wikipedia contributors. (2020) Buck converter — Wikipedia, the free encyclopedia. [Online; accessed 24-August-2020]. [Online]. Available: https://en.wikipedia.org/w/index.phptitle=Buck$_$converter\&$oldid=970174022$

[11] Williams and Wang LLC. Hc-sr04 ultrasonic distance sensor. [Online; accessed 24-August-2020]. [Online]. Available: https://www.bananarobotics.com/shop/HC-SR04-Ultrasonic-Distance-Sensor

[12] Rplidar a1. [Online; accessed 09-November-2020]. [Online]. Available: https://www.slamtec.com/en/Lidar/A1

[13] Logitech c270 webcam. [Online; accessed 30-August-2020]. [Online]. Available: https://www.logitech.com/en-us/product/hd-webcam-c270#specification-tabular

[14] (2020) Traxxas guide to batteries and chargers. [Online; accessed 24-August-2020]. [Online]. Available: https://traxxas.com/battery-basics

[15] Perceptinssr v-2.0 radar. [Online; accessed 09-November-2020]. [Online]. Available: https://www.perceptin.io/post/passive-perception-with-sonar-and-mmwave-radar

[16] A. Bhaumik. (2012) Open source robotics software platforms. [Online; accessed 24-August-2020]. [Online]. Available: https://www.opensourceforu.com/2012/03/open-source-robotics-software-platforms/

[17] Ros/introduction - ros wiki. [Online; accessed 24-August-2020]. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[18] Ros/concepts - ros wiki. [Online; accessed 24-August-2020]. [Online]. Available: http://wiki.ros.org/ROS/Concepts

[19] Master - ros wiki. [Online; accessed 24-August-2020]. [Online]. Available: http://wiki.ros.org/Master

[20] Nodes - ros wiki. [Online; accessed 24-August-2020]. [Online]. Available: http://wiki.ros.org/Nodes

[21] Messages - ros wiki. [Online; accessed 24-August-2020]. [Online]. Available: http://wiki.ros.org/Messages

[22] Topics - ros wiki. [Online; accessed 24-August-2020]. [Online]. Available: http://wiki.ros.org/Topics

[23] Wikipedia contributors, "Robot operating system — Wikipedia, the free encyclopedia," 2020, [Online; accessed 25-August-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Robot$_$Operating$_$System\newline&oldid=974664176

[24] rqtgraph - ros wiki. [Online; accessed 25-August-2020]. [Online]. Available: http://wiki.ros.org/rqt$_$graph

[25] Jetson.gpio github repositry. [Online; accessed 26-August-2020]. [Online]. Available: https://github.com/NVIDIA/jetson-gpio

[26] Adafruit servokit repository. [Online; accessed 29-August-2020]. [Online]. Available: https://github.com/JetsonHacksNano/ServoKit

[27] What is opencv? [Online; accessed 26-August-2020]. [Online]. Available: https://www.wisegeek.com/what-is-opencv.htm

[28] OpenCV contributors. (2020) Opencv. [Online; accessed 26-August-2020]. [Online]. Available: https://opencv.org/

[29] usbcam github repositry. [Online; accessed 26-August-2020]. [Online]. Available: https://github.com/ros-drivers/usb$_$cam

[30] rplidar github repositry. [Online; accessed 26-August-2020]. [Online]. Available: https://github.com/Slamtec/rplidar$_$ros

[31] radar github repositry. [Online; accessed 09-November-2020]. [Online]. Available: https://github.com/astuff/kvaser$_$interface

[32] Kangalow. Servo control using pwm driver. [Online; accessed 26-August-2020]. [Online]. Available: https://www.jetsonhacks.com/2019/07/22/jetson-nano-using-i2c/

[33] C. H. Walsh, "Localizing, modeling, and drifting an autonomous racecar," Master's thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 6 2018.

[34] A. Braden, D. Architect, and T. 4, "Development and implementation of autonomous racecar systems," *Beaverworks RACECAR*, 2016.

[35] Autonomous lane navigation via opencv. [Online; accessed 12-November-2020]. [Online]. Available: https://towardsdatascience.com/deeppicar-part-4-lane-following-via-opencv-737dd9e47c96