People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

MASTER

In **Electronics**

Options:**Computer Engineering**

Title:

Building a Self driving Car using ROS

Presented by:

- **AOUGACI ali**
- **KHACHOUCHE lokmane**

Supervisors:

Dr, H. BELAIDI

Registration Number:………/2019

ACKNOWLEDGEMENT

The completion of this project could not have been possible without the participation and assistance of so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged.

We would like to thank our supervisor, Dr. BELAIDI Hadjira, for her priceless guidance, encouragement and advice she provided throughout this tough final semester. She gave us all her time and support that have been extremely helpful.

Dedication

Every challenging work needs self-efforts as well as guidance of Elders those who were very close to our heart.

My humble effort I dedicate to my loving

***parents** and my **family members**, Whose affection, love, encouragement and prays of day and night make me able to get such success and honor.*

*Along with all **my friends**, **hardworking** and respected **Teachers.***

Abstract

This project consists of designing and building an autonomous mobile vehicle platform able to drive itself without human intervention. The appropriate hardware equipments and sensors suitable for the desired tasks and for the vehicle navigation are selected. Moreover, the car architecture is designed. Then, the Robot Operating System (ROS) is used for the software implementation, for sensors and actuators interfaces and for the approach execution. The robot Platform is a microprocessor and microcontroller based system.

TABLE OF CONTENTS

LIST OF FIGURES

Abbreviations

ROS	Robot Operating System
SW	Software
HW	Hardware
IMU	Inertial Measurement Unit
ECS	Electronic Speed Control
RC	Remote Controlled
SLAM	Simultaneous Localisation and Mapping
SSH	Secure Shell
URDF	Universal Robotic Description Format
IPC	inter-process communication
IP	based communication
AV	Autonomous vehicle
GPS	Global Positioning System
LiDAR	Light Detection and Ranging
PWM	Pulse Width Modulatio
RAM	Random Access Memory
ROM	Read Only Memory
USB	Universal Serial Bus

One such technique that can be used to create autonomous vehicles is the Robot Operating System (ROS). ROS is an operating method that includes a number of tools and libraries aimed at increasing the comprehensiveness of robotics programming. Furthermore, to that end, we can consider the autonomous cars as robots because they actually are: robots with sensors (GPS, odometers, ultrasound detectors etc.) and driven systems (tires, servos, engines...). ROS implements a range of robotic programming conferences.

However, despite advances, it is still very hard to program the robots. One reason is that robots can operate on a variety of software systems (SW) in distinct hardware (HW) settings. Reusability of code is a great problem because SW often is very interconnected and designed for a certain HW. Even if a piece of code is identified from another's job, a specific code reused

The initiative in particular is appealing as there is a rapid increase of common interest in self-driving cars. The autonomous driving community could be very helpful with the solutions, particularly those using open-source SW such as ROS.

I.1. Motivation

All these years passed studying electronics made us able to gain more knowledge about this field. Developing a self driving car with various sensors and capabilities is a challenge thatwe wanted to accomplish, it also leads to being able to mount additional sensors for whatever might be needed in future applications.

I.2. Goal of the Project

The ultimate objective of thisproject will be to construct an autonomous RC automotive prototype with ROS, which can operate on a plain ground. The car's primary entry is a camera shot,which is installed at the top, in real time. The system should then issue appropriate steering controls and regulate the vehicle. The aim of this project is to educate the vehicle how to navigate because the camera and ultrasonic sensor will be the only input for the engine. Its behaviour to avoid obstacles! Rent issue that can be solved, too,but it is beyond the reach of this project that we combine it with steering, producing a single control to deal with both situations. However, ultrasound devices will be used to identify barriers on the highway and prevent the vehicle deviation from the straight line. This works as a distinct module not interfering with the driving mechanism that is controlled by on-board computer.

Also to develop a list of requirements for the system and to design the structure of the system's components. After that to use the system to perform its features according to the developed system structure. The components arechosen in the secondchapter, by analysing their performances and capabilities according to the requirements.

An on-board computer will control the car. The vehicle will then befully independent of other machines.Another goal is to analyse how ROS can be used in the development of self-driving vehicles and to identify its potential benefits and challenges.

I.3. Background and Related Work

I.3.1.History of autonomous cars

The first self-driving car was produced by Norman Bel Geddesin General-Motors (GM's) 1939 exhibit.The car was an electric vehicle guided by radio-controlled electromagnetic fields generated with magnetized metal spikes embedded in the roadway. This car remained just as a concept until 1958 where it became a reality by General Motors. The car's front end was embedded with sensors called pick-up coils that could detect the current flowing through a wire embedded in the road. The current could be manipulated to tell the vehicle to move the steering wheel left or right [1].

In 1977, the Japanese improved upon this idea, using a camera system that relayed data to a computer to process images of the road. However, this vehicle could only travel at speeds below 20 mph. Improvement came from the Germans a decade later in the form of the VaMoRs, a vehicle outfitted with cameras that could drive itself safely at 56 mph. As technology improved, so did self-driving vehicles' ability to detect and react to their environment [1].

I.3.2.Autonomous cars today

Self-driving autonomous vehicles have arrived. Utilizing technologies such as radar, GPS, 360-degree camera systems and powerful onboard processing computers, driverless vehicles will eventually be rolled out to many industries including fleet, long-haul trucking, livery, Uber and on-demand car services. The driverless social paradigm shift is fast approaching.

At present, many vehicles on the road are considered to be semi-autonomous due to safety features like assisted parking and braking systems, and a few have the capability to drive, steer, brake, and park themselves. Autonomous vehicle technology relies on GPS capabilities as well as advanced sensing systems that can detect lane boundaries, signs and signals, and unexpected obstacles.

Autonomous vehicles are expected to bring with them a few different benefits, but the most important one is likely to be improved safety on the roads. The number of accidents caused by impaired driving is likely to drop significantly, as cars can't get drunk or high like human drivers can [2].

I.3.3.The idea of self-driving car

A vehicle that can operate autonomously should be prepared to ride without human feedback. In order to accomplish this, the independent vehicle must feel, navigate and respond without communication with human beings. The vehicles themselves can view their environment using a broad variety of devices including LIDar, RADAR, GPS, wheel odometry devices and cameras. In fact, the autonomous car needs a monitoring scheme capable of understanding sensor information and of making a distinction between road signs, barriers, peat busses and other anticipated and unexpected road events.

It needs to meet at least three significant functions in order to be known as a robot: sensing, planning, and acting. To call a vehicle self-employed, the same conditions should be satisfied[3].Self driving vehicles are mainly robot vehicles, which can decide on how to get from A to B. The driver must only indicate the location and it should be safe for the independent vehicle to bring him or her there. To turn a normal vehicle into a self-service vehicle, sensors and a built-in laptop are needed.

A car must work closely together to autonomously run a number of real-time systems. Include environment mapping and comprehension, location, trajectory scheduling and motion control as recognized by [4]. In order to have a platform to function on these real-time systems, the automatic driving car requires to be fitted with the suitable sensors, computer Hardware (HW), networking and Software (SW) facilities.

I.3.4.Vehicle autonomy levels

The Society of Automotive Engineers (SAE) defines 6 levels of driving automation ranging from 0 (fully manual) to 5 (fully autonomous). These levels have been adopted by the U.S. Department of Transportation [5]. These 6 levels are explained in Table 1.

Table 1: Autonomous Driving - Levels of Automation

Level	Automation	System
Level Zero	No Automation	The rider does all the main duties such as driving, braking, speeding or slowing, etc.
Level One	Driver Assistance	The car can support secondary and tertiary functions, but the driver still carries out all the major duties and tracking the environment.
Level Two	Partial Automation	The car can support driving or accelerating functions and can disengage the driver from certain duties. The driver must always be prepared to manage the car and remain accountable for the most critical safety tasks and environmental tracking.
Level Three	Conditional Automation	The car itself is responsible for all environmental monitoring (using sensors such as LIDAR). At this level the driver's attention continues to be critical but can decrease "safety-critical" functions like braking and leave the driver behind in the safe conditions
Level Four	High Automation	The car can steer, brake, speed, monitor the car and the road as well as react to incidents, determine when routes are changed, turned and signals are used
Level Five	Complete Automation	This amount of autonomous driving does not require any human attention. No pedals, brakes or wheels are required as the autonomous car model monitors all critical duties, environmental monitoring and identifies distinctive riding circumstances, such as traffic jams.

I.3.5. Sensors of self-driving vehicles

a) Perception Sensors

One of the most important tasks of autonomous systems is to acquire knowledge about its environment. This is done by taking measurements using various sensors and then developing inferences from those measurements.

An autonomous vehicle may experience unforeseen events on the road which it needs to register and act accordingly. Discant et al[6] present a brief study about the available sensors for obstacle detection. According to them perception sensors are classified into two types: active and passive sensors.

Active sensors emit their own energy into the environment, then measure the environmental reaction. For object detection, few different types of active sensors can be used: Radar, LIDAR, SONAR, Time-of-flight (TOF) camera [6].

Passive sensors measure ambient environmental energy entering the sensor. Examples of passive sensors include temperature probes, microphones, visible spectrum cameras and infrared cameras. Visible spectrum cameras can be further subdivided into monocular camera and stereo camera [6].

b) RequirementSensors

According to [7], the following sensors, shown in figure 1.1, should be present in all self-driving cars:

- ***Global positioning system (GPS):***Global positioning system is used to determine the position of a self-driving car by triangulating signals received from GPS satellites [7]. It is often used in combination with data gathered from an IMU and wheel odometry encoder for more accurate vehicle positioning and state using sensor fusion algorithms. The general form of GPS is shown in figure 1.2.

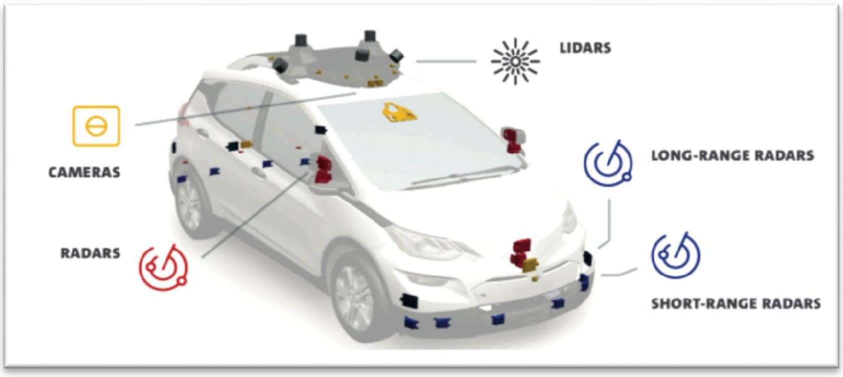


Figure 1.1.Self-driving car sensors.

- ***Light detection and ranging (LIDAR):***A core sensor of a self-driving car, this measures the distance to an object by sending a laser signal and receiving its reflection. It can provide accurate 3D data of the environment, computed from each received laser signal. Self-driving vehicles use LIDAR to map the environment and detect and avoid obstacles.

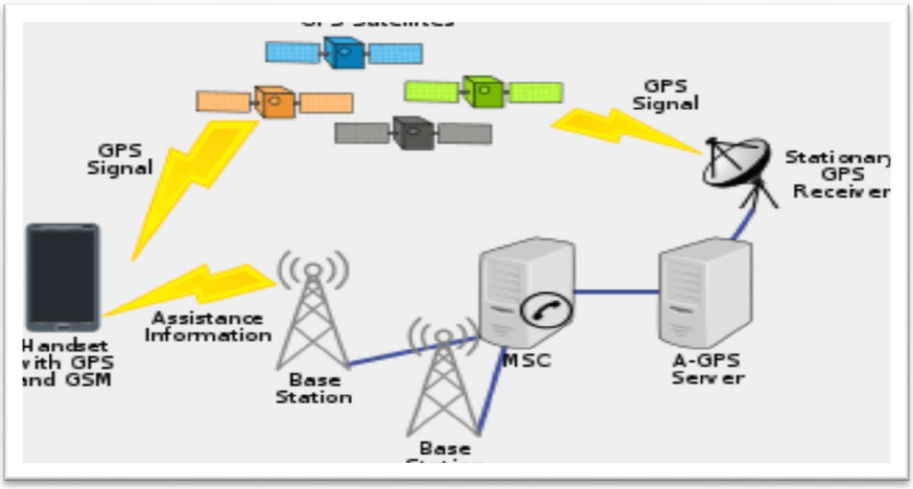


Figure 1.2. General form of GPS

- ***Camera:***Camera on board of a self-driving car is used to detect traffic signs, traffic lights, pedestrians, etc... by using image processing algorithms (see figure 1.3).

Figure 1.3. SainSmart Wide Angle Fish-Eye Camera Lenses

- ***RADAR:***RADAR is used for the same purposes as LIDAR. The advantages of RADAR over LIDAR are that it is lighter and has the capability to operate in different conditions. While it has longer range, all RADAR categories have a limited field of vision.
- ***Ultrasound sensors:***Ultrasound sensors play an important role in the parking of selfdriving vehicles and avoiding and detecting obstacles in blind spots, as their range is usually up to 10 metres (see figure 1.4).

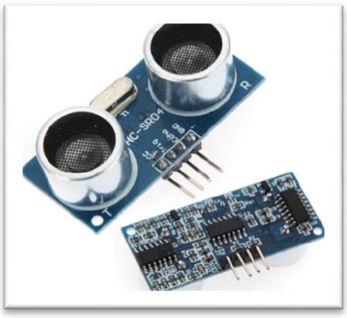


Figure 1.4. HC-SR04 Ultrasonic sensor

- ***Wheel odometry encoder:***Wheel encoders provide data about the rotation of car's wheels per second. Odometry makes use of this data, calculates the speed, and estimates the car's position and velocity based on it (shown in figure 1.5).Odometry is often used with other sensor's data to determine a car's position more accurately.

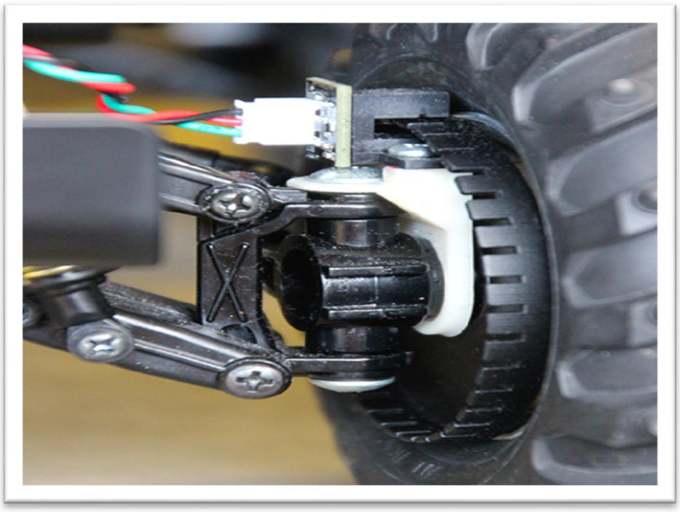


Figure 1.5. Wheel odometry encoder

- ***Inertial measurement unit (IMU):***An IMU, illustrated in figure 1.6, consists of gyroscopes and accelerometers, with one pair oriented towards each of the axes. These sensors provide data on the rotational and linear motion of the car, which is then used to calculate the motion and position of the vehicle regardless of speed or any type of signal obstruction.

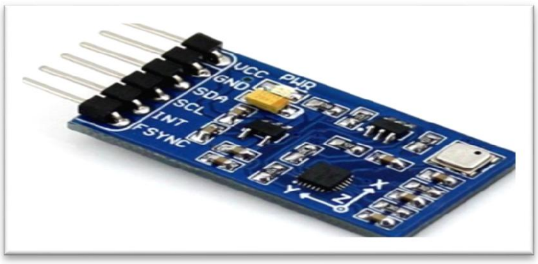


Figure 1.6. Inertial measurement unit (IMU)

- ***On-board computer:***This is the core part of any self-driving car. As any computer, it can be of varying power, depending on how much sensor data it has to process and how efficient it needs to be. All sensors are connected to this computer, which has to make use of sensor's data by understanding it, planning the route and controlling the car's actuators. The control is performed by sending the control commands such as steering angle, throttle and braking to the wheels, motors and servo of the autonomous car.

I.4. SW block diagram of self-driving vehicles

Figure 1.7 illustrates the SW block diagram of the standard self-driving car, as presented by[8].

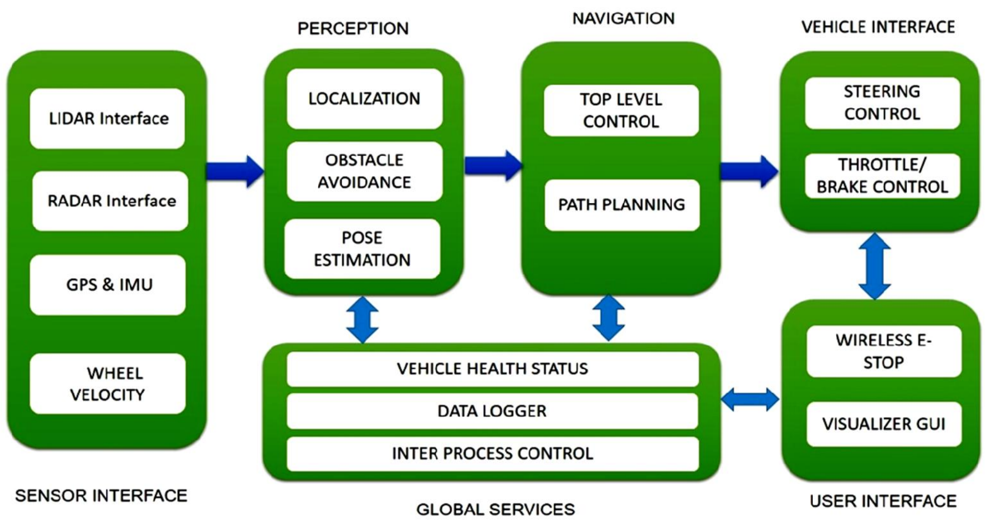


Figure 1.7. SW block diagram of a self-driving car.

Each block can interact with others using the inter-process communication (IPC) or shared memory. In this scenario, ROS message middleware is a perfect fit. They introduced a system of publishing and subscribing to do these tasks in the DARPA Challenge. Lightweight Communications and Marshalling were one of the MIT library growth challenges for 2006 in DARPA.

- **Sensor interface modules**: As the module name suggests, this block contains all the communication between the sensors and the car. The block allows us to supply all other blocks with different sensor data. LIDAR, camera, radar, GPS, IMU and wheel encoders are the most common devices.
- **Perception modules**: These modules process sensor information such as LIDAR, camera, radar, and section information to detect moving and stationary items. They also assist to identify the auto drive compared to the environmental digital map.
- **Navigation modules**: The behaviour of the automobile is determined by this module. It has movement planners for multiple conduct in the robot and the finite state machines.
- **Vehicle interface**: The control instructions such as steering, throttle and brake control are sent to the car via a Drive-by-Wire (DBW) interface after the route scheduling. DBW operates essentially via the CAN bus. The DBW interface is only supported by certain cars. Examples include the Lincoln MKZ, VW Wagon Passat and a few Nissan designs.
- **User interface**: we can display a map and specify the target on a touch screen. An emergency stop key is also available for the customer.
- **Global services**: This module helps to record the information and has time and message pass-over help for the reliability of the software.

I.5. Robot Operating System

ROS is not an actual operating system, but rather a meta-operating system. Simply put, ROS works on top of other operating system and allows different processes to communicate with each other during runtime. As a meta-operating system, ROS offers a communications layer, in a structured manner, running on top of the operating systems of host computers[9]. Usage of ROS is not limited to robotics only, as majority of ROS tools are compatible with peripheral hardware and can be used for various purposes. ROS core consists of more than two thousand packages, where each package has its specific functionality.

Hence, ROS is a set of tools that provides the functionality and services of an operating system on a single, or over multiple computers. These services include abstraction of the hardware, exchange of messages between processes, management of packages, etc.... Ademovic [10]argues that ROS's greatest strength is the number of available ROS tools.

One very important characteristic of ROS is that it is completely open source. ROS was designed with the goal of robotic SW reusability in mind. It is stated by Quigley[9] that writing SW for robots is difficult, particularly as the scale and scope of robotics grows. Different types of robots can have widely varying HW, making code reuse extremely challenging. Further, robot SW is tightly coupled, and the extraction of reusable code can be very difficult and ROS is built to overcome these problems [9].

To explain ROS quigley summarised the philosophical goals of ROS using the following five statements:

***ROS is peer-to-peer*:**ROS nodes are units of execution that communicate with each other directly or via publish/subscribe mechanisms.

***ROS is tools-based*:** It uses a microkernel design and several small tools and modules.

***ROS is multilingual*:**It has support for C++, Python, Octave and LISP.

***ROS is 'thin'*:** It uses a catkin build system to provide code segmentation in terms of packages and libraries.

***ROS is free and open source*:** ROS is publicly available under a BSD license.

I.5.1.ROS Overview

ROS uses internet protocol (IP)-based communication to transfer data between ROS nodes. This way, ROS splits the robotic SW into ROS nodes that can be executed on one machine or on the distributed computer cluster. ROS nodes use publish/subscribe channels to exchange information amongst themselves, but they can also provide callable services to other nodes. A running ROS system has one master node (roscore) that acts as a name server and allows other nodes to find each other to form direct connections [11].This architecture results in very low coupling between nodes and promotes their reuse. For example, the same ROS nodes can be used without modification in both the actual robot and in the simulator.

The fundamental concepts of ROS implementation are *nodes*, *messages*, *topics* and *services*. Figure 1.8 illustrates the role of the ROS master node: roscore, which is the essential part of each ROS-based program. Simply put, roscore is a set of core ROS nodes that are essential for ROS-based application to be able to run [11]. The roscore master node must be running for ROS nodes to communicate[11]. When roscore is active, nodes can exchange messages by publishing and subscribing to certain topics or by directly invoking the services and actions of the other nodes.

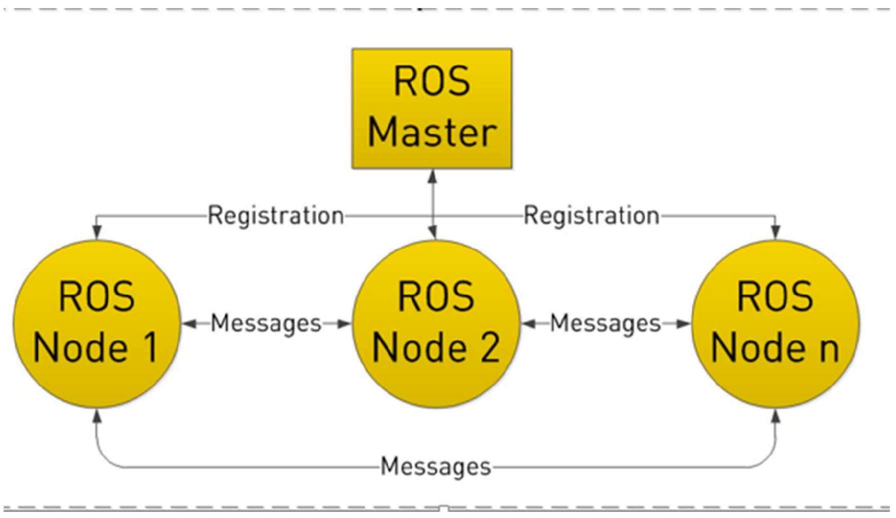


Figure 1.8. Illustration of ROS nodes and messages [11]

Figure 1.9 illustrates the ROS concepts (nodes, messages and topics) and how they correlate. Services, a way of communicating between nodes, do not use publish/subscribe mechanism, but rather directly invoke the services of the other node.

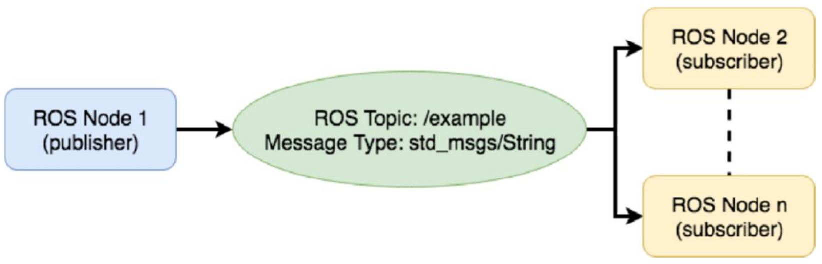


Figure 1.9. Visualisation of ROS concepts [11]

As identified in the article written by Tellez[12],ROS has two major drawbacks:

***Roscore is a single point of failure*:**As roscore must be constantly running for ROS program to run, roscore poses a security threat for all ROS-based programs. The rest of the system might be running smoothly and written perfectly, but if roscore terminates, the whole ROS-based program shuts down too.

***Security issue*:**The access to ROS network is not secured, which is a big security threat for autonomous car's which are using ROS. The communication among ROS nodes is not secured, thus the whole system is vulnerable. Someone who gains the access to car's ROS network can access and alter the car's behaviour.

Tellez [12] also stated, however, that both drawbacks are being addressed in the newest version of ROS—ROS version 2. Even with the present negatives, it can be argued that ROS is a good solution for developing autonomous driving.

The ROS core concepts—nodes, topics, messages and services—are explained in detail in the subsections below.

I.5.2. ROS nodes

A node is a process that performs computation and it can be seen as a single unit of execution in the ROS ecosystem. Nodes can communicate with each other using client server–like architecture, where each node is assigned a specific task and can serve both as a client and a server at the same time and [11]. Nodes should perform their own tasks and communicate their results with the other nodes. The significant advantage offered by this architecture is fault tolerance (as each node is an isolated part of the system).

I.5.3. ROS messages

ROS provides over 200 predefined messages and the ability toROS creates custom ones. Messages are exchanged between ROS nodes using publish/subscribe mechanism. One ROS node would publish the ROS message to certain ROS topic, while the other ROS node would subscribe to that ROS topic and obtain the sent ROS message. ROS Messages are typically described in text files inside *msgs* folder under ROS folder structure. These text files are following certain standards for description of ROS messages. The description format of ROS messages is fairly simple. Each ROS message is a data structure which contains primitive types (integers, floats or booleans) or an array of primitive types. Additionally, ROS message can contain the other ROS message or an array of ROS messages as a data type. ROS messages can be also exchanged in direct communication between nodes, called ROS Services and in this case, the messages should be inside of the *srv* folder [11].

I.5.4. ROS topics

ROS nodes communicate with each other over **topics**

- If we want to send messages, we**publish** to a topic

- If we want to receive messages, we**subscribe** to a topic

ROS topics are used when ROS nodes are communicating using publish/subscribe mechanism. Each ROS topic has a unique name, so that ROS nodes can publish or subscribe to it.

I.5.5. ROS services

When nodes need to communicate directly with each other, they use ROS services. In such case, publish/subscribe mechanism is evaded and nodes can communicate to each other directly using the defined request and reply messages. Even that, ROS services are increasing the system performance as they are form of direct communication.

I.6. Conclusion

This chapter was a deep discussion of Robot Operating System (ROS) and self-driving cars and their implementation. The chapter started by discussing the basics of self-driving car technology and its history. Afterward, we discussed the core blocks of a typical self-driving car. We also discussed the concept of autonomy levels in self-driving cars. Then, we took a look at different sensors and components commonly used in a self-driving car. After that, we introduced Robot Operating System (ROS) and we take a general look about its basics (nodes, messages, topics, services).

Chapter 2: Hardware System Design

After chapter 1, where a deep discussion about ROS and self driving theory were given, in this chapter the necessary hardware requirements will be discussed. The steps followed to build our RC Car from empty chassis to a ready self driving car will be detailed within this chapter. Our RC car will be implemented in such a way that it can be controlled by agent via keyboard and driven by itself avoiding obstacles.Each component will be described alone then how it is connected to the whole circuit. At the end of this chapter, the complete circuit diagram and the final look of the car will be represented.

II.1.System Requirements

The system needs the following characteristics to guarantee effective implementation:

- The device requires to be able to obtain and perform commands from the on board computer that runs the engine in the event of manual drives and is linked to a servo and regulates its steering direction with the help of libraries. And also by using servo and a servo control system to do this.
- The car must have a camera mounted in front which will be connected to the onboard computer.
- It needs to capture images from the camera and send them in real time to the onboard computer.
- It needs to read data from the sensors (ultrasonic, IMU …) and send it to in real time to the onboard computer.

II.2. System Hardware

The elements that crossed the hardware and software of the car are the most significant parts of the self-driving RC Car: The Raspberry Pi 3 board and Arduino Uno board. Raspberry Pi 3 is an on-board microprocessor, while Arduino is an on-board microcontroller.

II.2.1.Raspberry Pi 3 Model B

As computation and processing unit, a robot can have either a computer or a microcontroller. In case the robot has cameras, laser-scans and LIDARs;so, powerful computers are needed and used to handle information.

Chapter 2: Hardware System Design

A microprocessors' panel is the Raspberry Pi 3, shown in figure 2.1. It is an integrated panel and a single panel computer that can be used as a normal PC to load and operate a working scheme.It has a chip scheme comprising ARM, RAM, GPU and all conventional software ports as parts. For our RC car,it operates ROS on Ubuntu 16.04. Its characteristics are summarized in table 2.1.

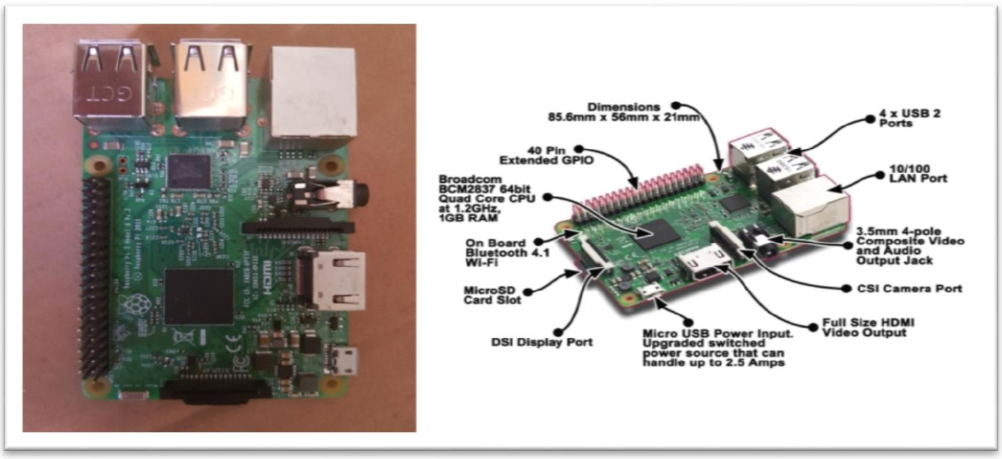

Dimensions 85.6mm x 56mm x 21mm
4 x USB 2 Ports
40 Pin Extended GPIO
10/100 LAN Port
Broadcom BCM2837 64bit Quad Core CPU at 1.2GHz, 1GB RAM
On Board Bluetooth 4.1 Wi-Fi
3.5mm 4-pole Composite Video and Audio Output Jack
MicroSD Card Slot
CSI Camera Port
Full Size HDMI Video Output
DSI Display Port
Micro USB Power Input. Upgraded switched power source that can handle up to 2.5 Amps

Figure 2.1. Raspberry Pi 3 Model B.

TABLE 2: Characteristics of Raspberry Pi 3 Model B

Parameter	Characteristics
Dimensions	85.60 mm × 56 mm × 21 mm
Weight	45 g. (excl. case)
System on a Chip	Broadcom BCM2835
Processor	ARM1176JZFS (700 MHz, option for overclocking up to 1000 MHz)
Video core	4 GPU
RAM	512 MB
Storage	4 GB SD card (max. 32 GB)
Interfaces	USB (2×), SD card, HDMI, 3.5 mm audio jack, GPIO (26 dedicated pins)
Network	10/100 Mbps wired Ethernet Wi-Fi USB Dongle
Power supply	5 v micro USB power supply
Operating temperature range	−25 to +80 °C

II.2.2.Arduino Uno

Arduino Uno is an ATmega328P-based microcontroller panel. It contains 14 input / output ports (6 of which can be used as PWM exports), 6 inputs for analog purposes (including 16 MHz crystal) and 16 MHz quartz crystal. It includes everything necessary for the microcontroller to be connected to or powered by an AC-to-DC adapter or battery onto the laptop with the USB cable.

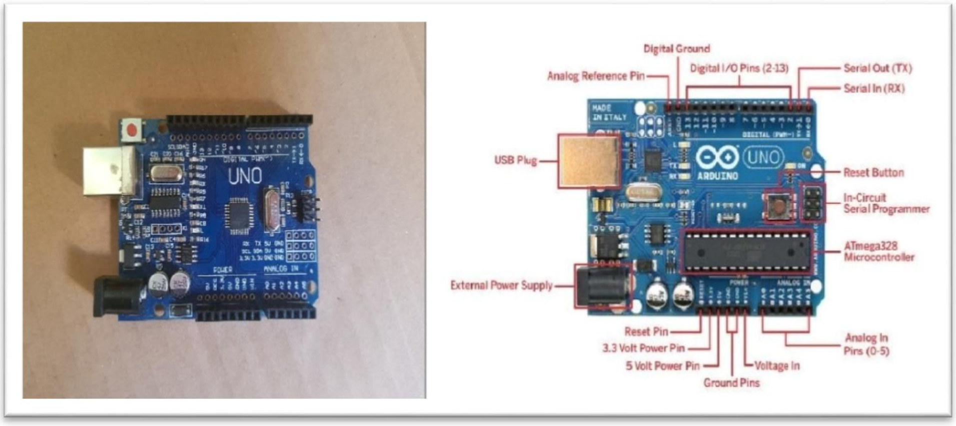


Figure 2.2. Arduino Uno board

An Arduino Uno panel is one of the most common integrated controller panels which are suitable for self-driving vehicles and used most frequently in robotics. To collect car position and location information, RC Car simulator was installed on the top of the RC Car's Raspberry Pi 3 with the help of an Adafruit 16-channel Modulation Pulsing Width (PWM) / Servo HAT module for Raspberry Pi 3.

II.2.3. RC Car chassis

This RC car was constructed from scratch because we installed all mechanical and electronic parts, components and sensors on the frame. Its size is suitable to carry all components.

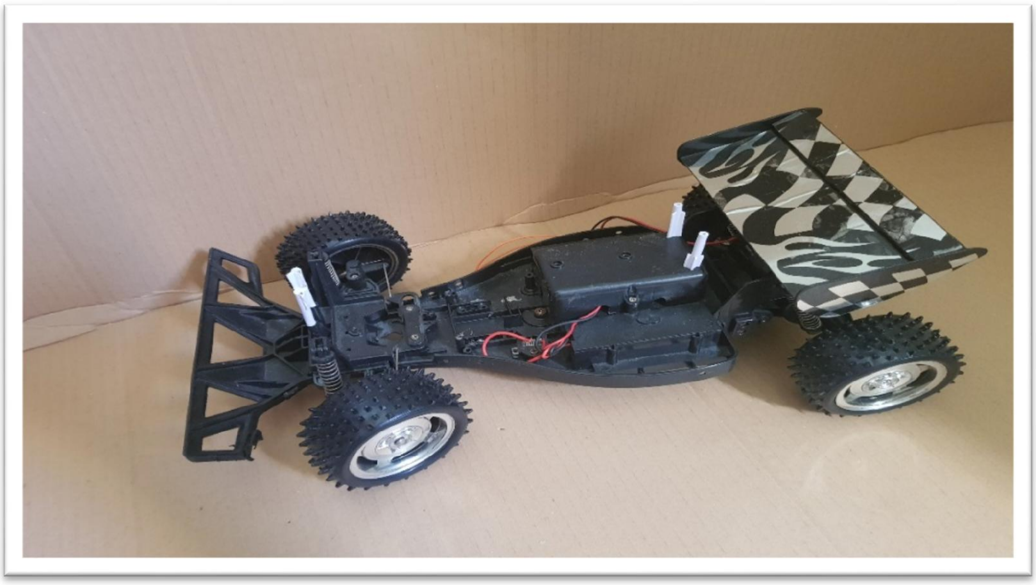


Figure 2.3. Simple RC Car chassis.

II.2.4. Micro servos

In RC car, the servo motor is used to regulate the vehicle wheels because it generates the torque needed to travel.To prevent snapping or collapsing, steering is vital to change the turning angle and orientation of the car. Servomotor is used for controlling the steering angle and ESC is used for controlling the velocity of the tires. A servomotor is shown in figure 2.4.

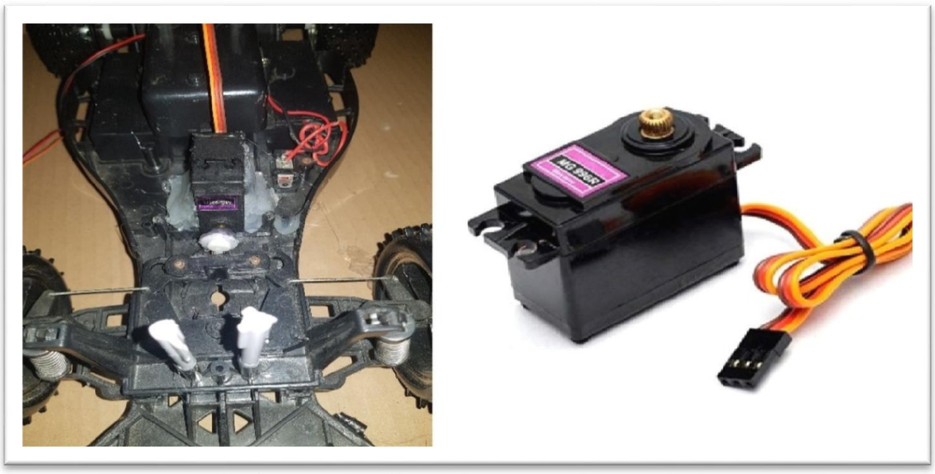


Figure 2.4. Servo Motor connection.

II.2.5. ESC (Electronic Speed Control)

We use an electrical speed controller ECS to control the speed of the dc motor using NPN transistor TIP29C, Which can switch up to 100V.

When PWMing a transistor, it is similar to pulsing an LED. The higher the PWM value, the faster the motor will spin. The lower the value, the slower it will spin. The ESC circuit is shown in figure 2.5.

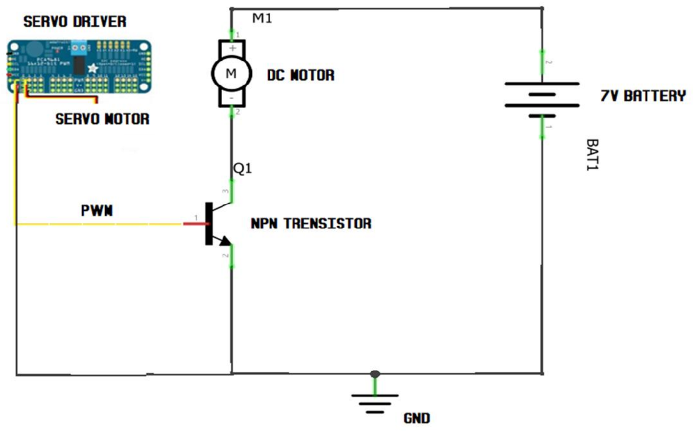


Figure 2.5.Circuit diagram of ESC.

II.2.6. Batteries and other required equipments

The ESC is powered from the 700mAh RC car's battery. All other components, mainly the Pi, need to be connected to an external source of power. A 2600mAh Power Bank is used to supply power to all our hardware and that should be enough for testing the car. The size and capacity of the battery depends on the size of the car and how long does the user want to drive it. Together with the two Batteries, a set of jumper wires is needed to connect the PCA9685 to the Raspberry Pi and to connect Arduino with IMU also to connect ultrasonic sensor to Raspberry Pi.

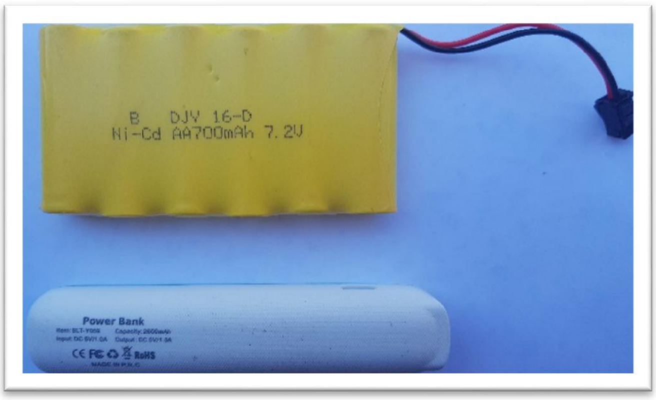


Figure 2.6. 2600mAh Power Bank and 700mAh Battery

A breadboard is also needed so that a voltage divider can be built which is needed because the echo in ultrasonic sensors will return 5v and that can damage GPIO on Raspberry Pi. The voltage divider connection is shown in figure 2.7.

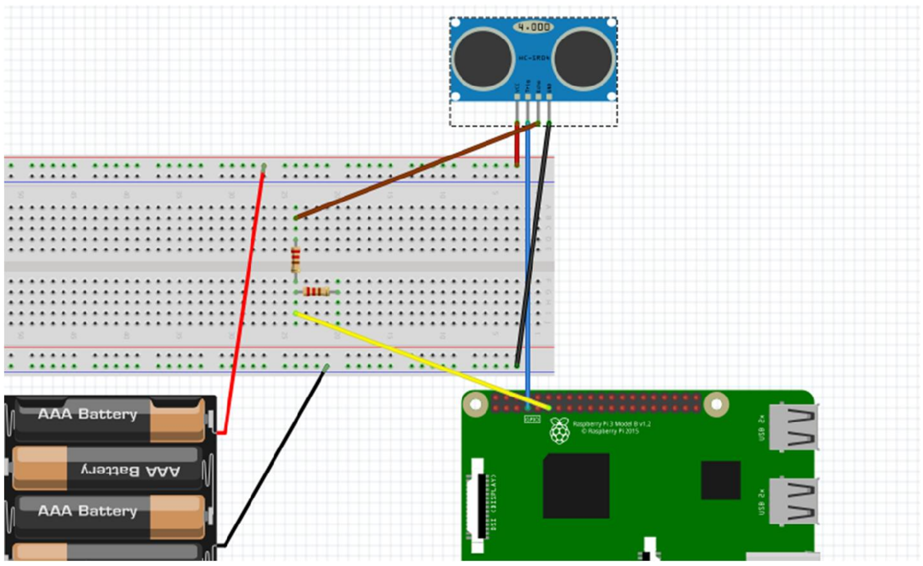


Figure 2.7. Voltage divider connection.

II.2.7. Camera

The needed information from the real environment to the autonomous car is delivered by an HD camera together with an ultrasonic sensor. This vehicle can securely and intelligently reach the target, thus avoiding the danger of human mistakes andobstacles. It also enables color identification in order to implement traffic light (not implemented in our project).

The 5MP camera module is ideal for tiny, space-saving Raspberry Pi initiatives.

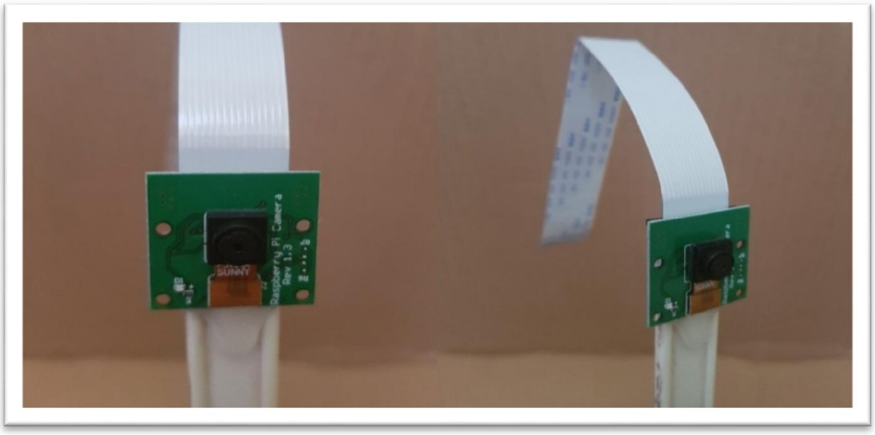


Figure 2.8. Raspberry Pi 3 Model B Camera

II.2.8.Ultrasound sensors

Three ultrasound range detectors HC-SR04s were used for our self-driving RC Car. As shown in figure 2.9, these ultrasound detectors were positioned to obtain the highest coverage of obstruction. Since the vehicle could progress with a maximum steering angle of 45 degrees, experimental tests demontrate that when three ultrasound devices were in use the optimum positions were displayed.

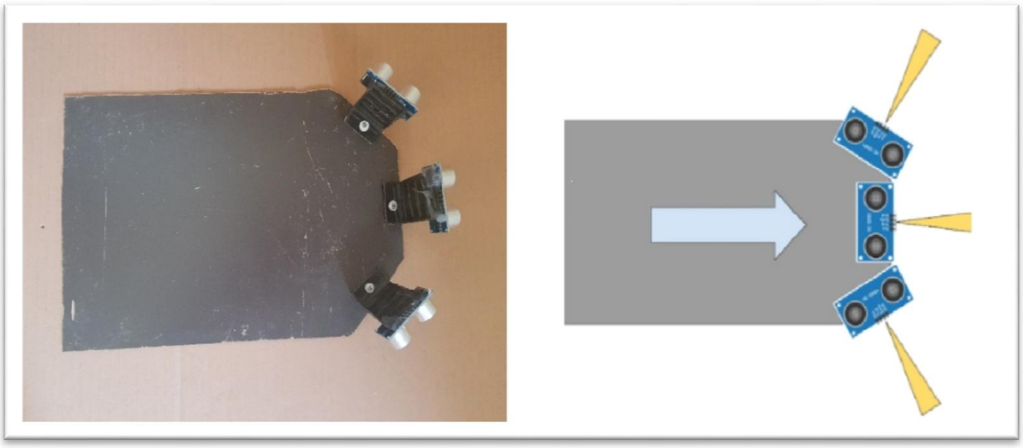


Figure 2.9. RC Car's ultrasound sensor placement.

The ultrasound sensors are connected with Raspberry Pi as shown in figure 2.10; each sensor has four pins Echo and Trig also GND and VCC. With, the Echo pin is the input and Trig is the output.

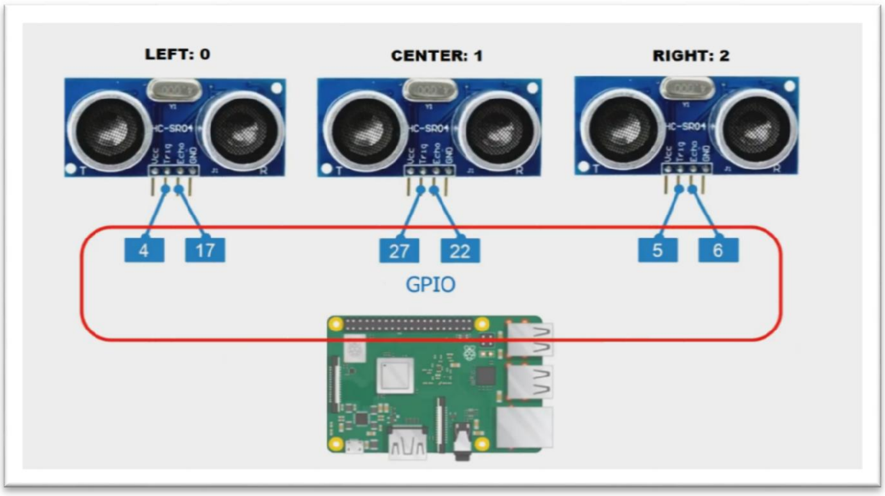


Figure 2.10. Ultrasound sensors connection with Raspberry Pi.

The Ultrasonic Sensor delivers out a high-frequency sound pulse and then calculates the time spendby the echo of the sound to reflect back. Then, the distance between the car and the obstacle is calculated with the next formula after the reverse signal the echo tag starts and the time ends(as illustrated in figure 2.11).

DISTANCE= TIME * (SPEED OF SOUND / 2)

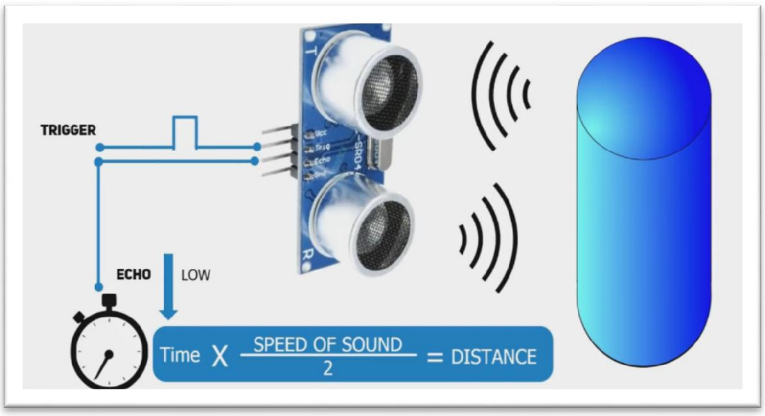


Figure 2.11. ultrasound sensors functioning.

II.2.9. IMU Integration

InertialMeasurement units (IMUs) are as essential for autonomous car as cameras and radars. An IMU is a tool that measures straight three linear speed elements and three rotational speed elements of the vehicle. An IMU is distinctive among the detectors discovered typically in anautonomous vehicle, because an IMU does not require an internal link and understanding the external world[11].

The IMU helps provide “localization” data. Software implements driving functions then combines this information with map and “perception stack” data that tell the car about objects and features around it.

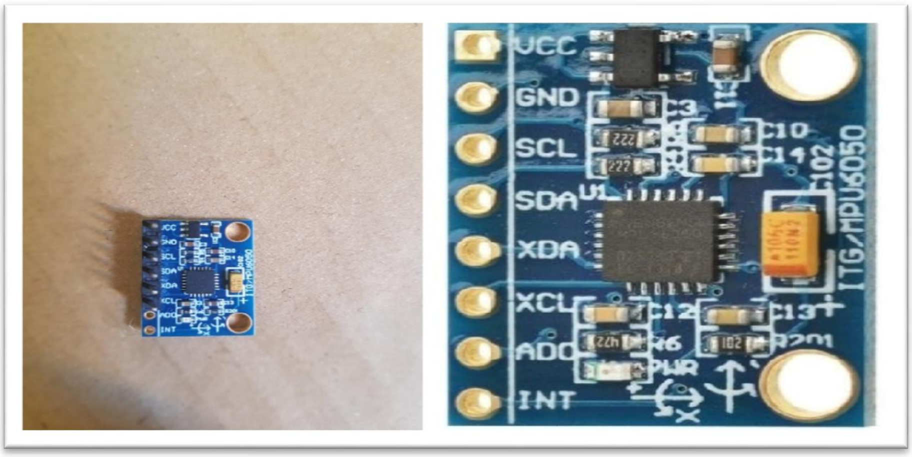


Figure 2.12: Inertial measurement unit (IMU).

The IMU is connected to the Arduino as shown in the diagram given in figure 2.13. If the MPU 6050 module has a 5V pin, then we can connect it to our Arduino's 5V pin. If not, we have to connect it to the 3.3V pin. The GND of the Arduino is connected to the GND of the MPU 6050 and the SCL to A5, SDA to A4 and INT to Pin 2.

Through I2C, the MPU 6050 communicates with the Arduino. And it is connected to the Arduino as shown in figure 2.13, we can connect the 5V pin toour Arduino. The arduino GND is then connected with MPU 6050 GND and A5 with SDA, A4 and Pin 2 with INT.

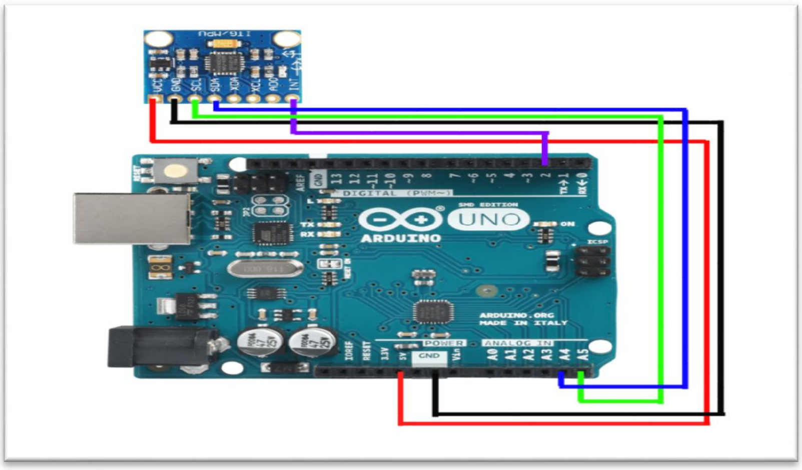


Figure 2.13: Interfacing IMU with Arduino Uno using fritzing.

II.2.10. Servo Controller

We need particular PWM signals to regulate the DC motor and servos. We chose to use the Adafruit PCA968516 Channel 12 Bit PWM Servo Driver (SD) as unique add-on board for the Pi. The Adafruit PCA9685 has been encouraged and can regulate up to 4 servo devices with complete PWM speed control. It is quite small and relatively simple to set up and comes with a computer library which is openly accessible.

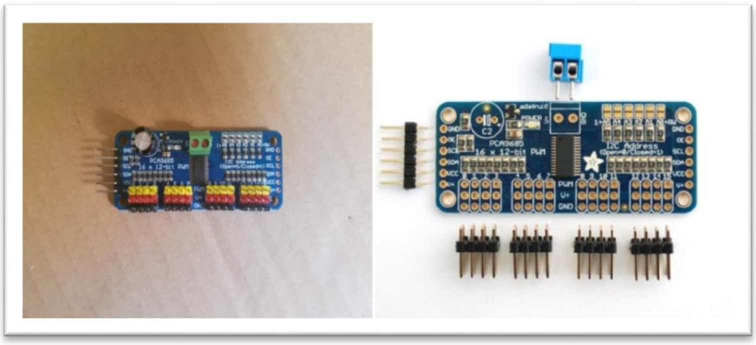
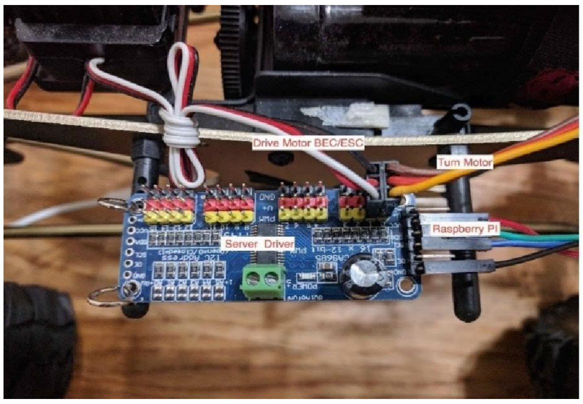
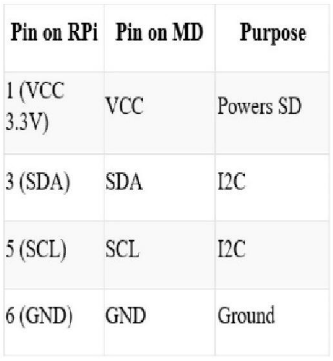


Figure 2.14: Adafruit PCA9685 16 Channel 12 Bit PWM Servo Driver.

Four GPIO pins from RPi need to be connected to the servo driver side pins (check figure Figure 2.15). The I2C protocol can be used to control this specific SD. The pin links are listed below.





Pin on RPi	Pin on MD	Purpose
1 (VCC 3.3V)	VCC	Powers SD
3 (SDA)	SDA	I2C
5 (SCL)	SCL	I2C
6 (GND)	GND	Ground

Figure 2.15: Interfacing adafruit servo driver with ESC, servo motor and raspberry pi 3.

We allow unconnected V+ and OE pins on the SD. For powering the load, V+ lock is used. We do not have to use this tool, because we have already attached the three-pin adapter of ESC with V+. Then attach the RPi to our USB power supply (see Figure 2.16).

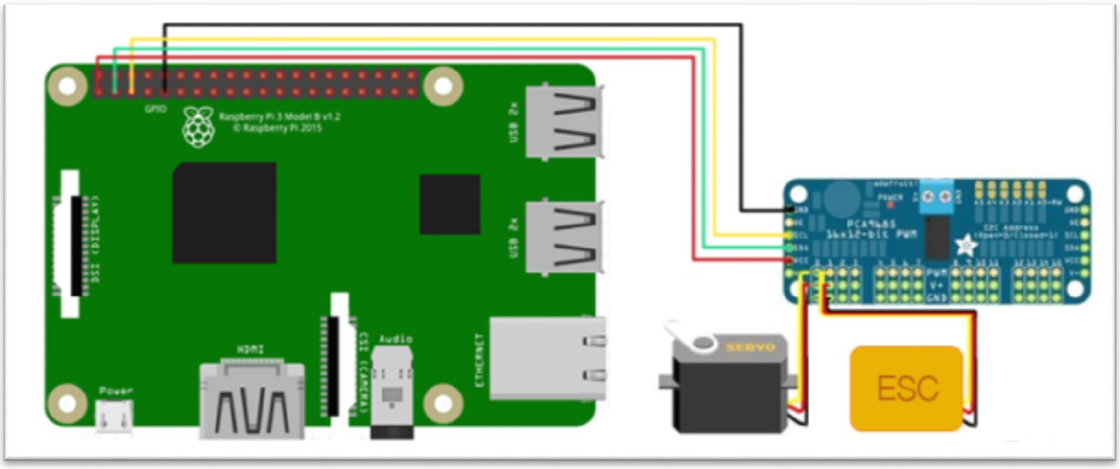


Figure 2.16: Electronic schema of interfacing adafruit servo driver with raspberry pi 3

II.3. Overall System Design

After we have collected all components and assembled each element in the pin and in the vehicle, we attach everything according to figures 2.17 and 2.18 bellows, which demonstrate our RC car's circuit diagram and final look. The general scheme shows how all components are connected.

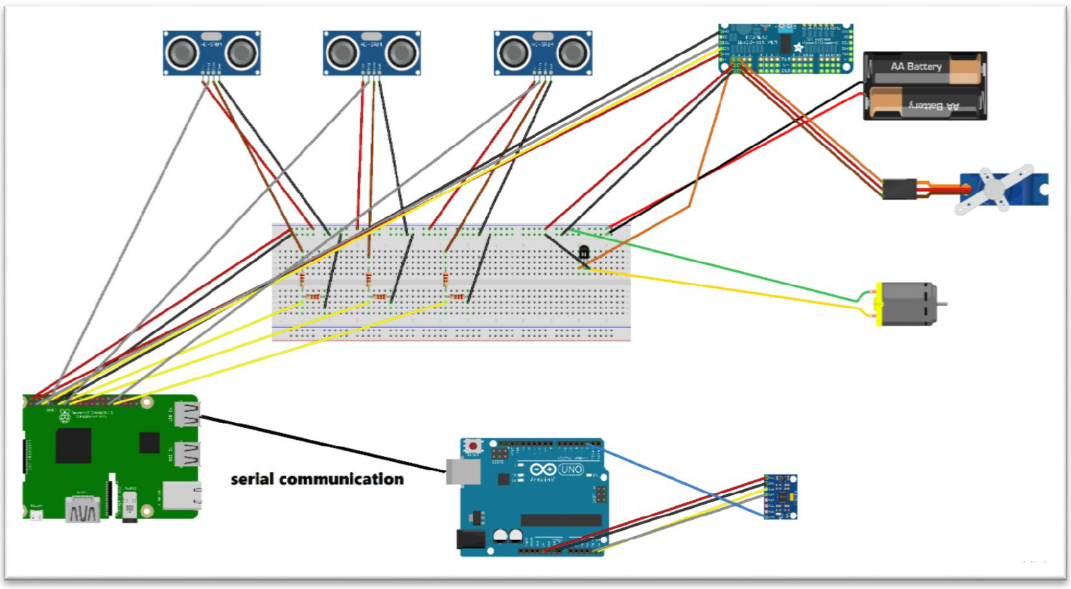


Figure 2.17: Circuit diagram of the whole RC car.

Figure 2.18illustrates the final look of our RC Car, with its main HW components marked.Here, the position and placement of the main HW components are visible.

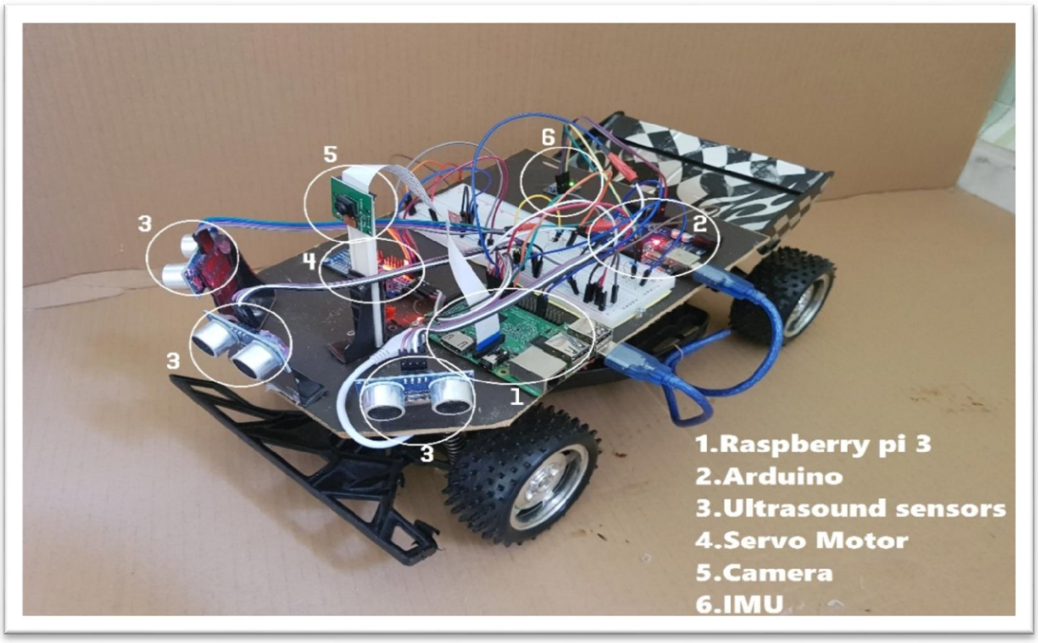


Figure 2.18. Final look of our RC Car with marked HW.

II.4. Conclusion

In this section, each portion of our self-sufficient vehicle is implemented in detail step by step. The scheme has been provided and explains all needed settings and layout.

The hardware system development is carried out at this stage. The remaining thing is the development of the software system described in the next chapter.

After the implementation of the HW in chapter 2 now we move to the SW implementation. First we will represent the operating system that we worked with; then, we will discuss the whole ROS architecture for our RC Car after that we go through details step by step.We will also establish serial connection between arduino and Raspberry pi using Rosserial.

III.1. The system Network

The Ubiquity Robotics Raspberry Pi image which is based on Ubuntu 16.04 operating System was installed and used in Raspberry Pi. It is pre-installed with ROS, and perfect for building Raspberry Pi robots; based on the wonderful work of Ubuntu Pi Flavor Markers. When the Raspberry Pi boots for the first time, it creates a Wifi access point which will be connected to our PC.

Figure 3.1 shows the network that we create and connect the PC and Raspberry pi to it to be able to establishSSH connection to control the RC Car.

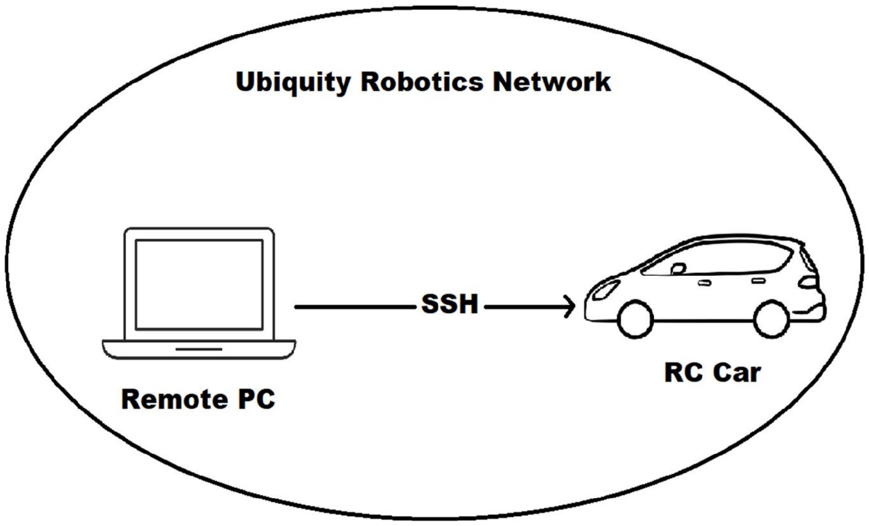


Figure 3.1. Network setup for development and control of the RC Car.

Secure Shell (SSH)is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line login and remote command execution, but any network service can be secured with SSH [13].

III.2.ROS Tools

ROS has a variety of GUI (Graphical User Interface) and command-line tools to inspect and debug messagesso we are going to use the following tools :

III.2.1. Gazebo

Gazebo is a dynamic robotic simulator with a wide variety of robot models and extensive sensor support. The functionalities of Gazebo can be added via plugins. The sensor values can be accessed to ROS through topics, parameters and services. Gazebo can be used when our simulation needs full compatibility with ROS. Most of the robotics simulators are proprietary and expensive; if they canot be afforded, Gazebo can be directly used without any doubt [14].

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At our fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community [14]. Figure 3.2 gives an illustration of Gazebo simulator.

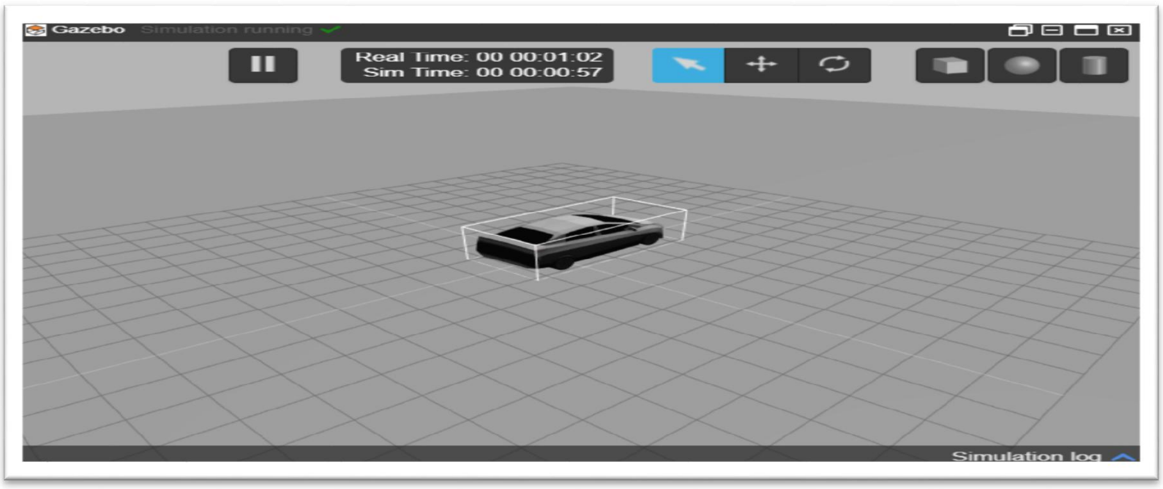
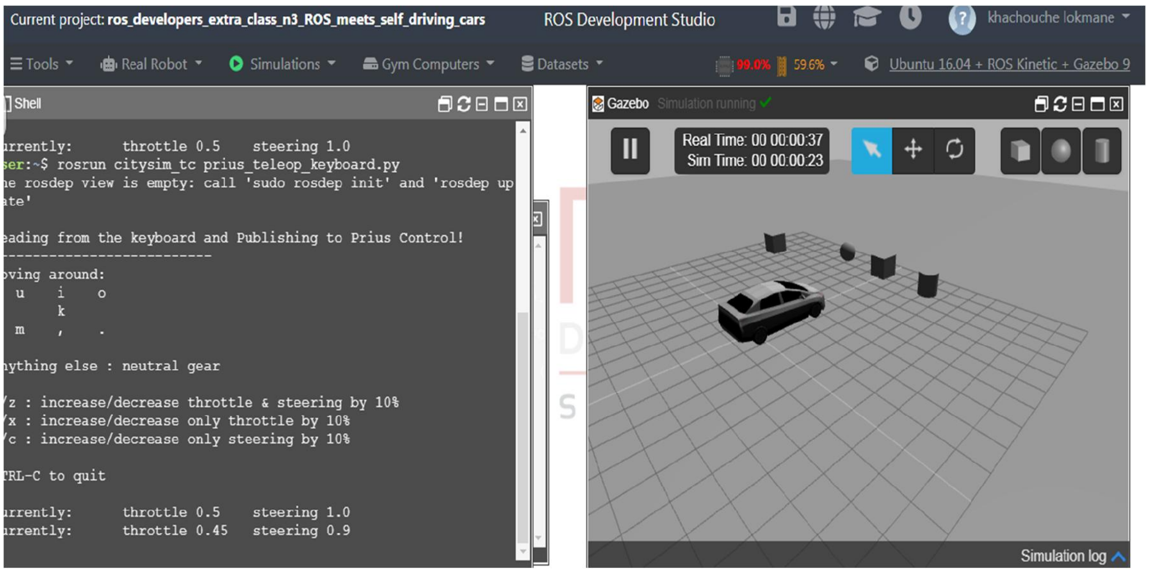


Figure 3.2. Gazebo simulator.

In ROS development process we use both HW implementation and simulation because some of the sensors was not acquired, and for that gazebo simulator was used as shown in figure 3..



```
urrently:      throttle 0.5    steering 1.0
ser:~$ rosrun citysim_tc prius_teleop_keyboard.py
he rosdep view is empty: call 'sudo rosdep init' and 'rosdep up
ate'

eading from the keyboard and Publishing to Prius Control!
--------------------------
oving around:
  u    i    o
       k
  m    ,    .

nything else : neutral gear

/z : increase/decrease throttle & steering by 10%
/x : increase/decrease only throttle by 10%
/c : increase/decrease only steering by 10%

TRL-C to quit

urrently:      throttle 0.5    steering 1.0
urrently:      throttle 0.45   steering 0.9
```

Figure 3.3. Model of our RC car simulation using gazebo.

III.2.2.Rviz

Rviz is one of the 3D visualizers available in ROS to visualize 2D and 3D values from ROS topics and parameters. Rviz helps visualizing data such as robot models, robot 3D transform data (TF), point cloud, laser and image data, and a variety of different sensor data[14]. This visualization will be used later for 3D model simulation of our RC Car.

III.3. ROS Implementation

In this section we will develop the ROS architecture for all self driving actuators and sensors, starting with the motor and servo than the ultrasonic sensors, camera, IMU and develop some autonomous car algorithms.

III.3.1. Manual control

In this section we will develop manual control system to control the RC car steering and speed of the motor using PWM, the command is coming from PC keyboard. The manual ROS architecture is shown in figure 3.4.

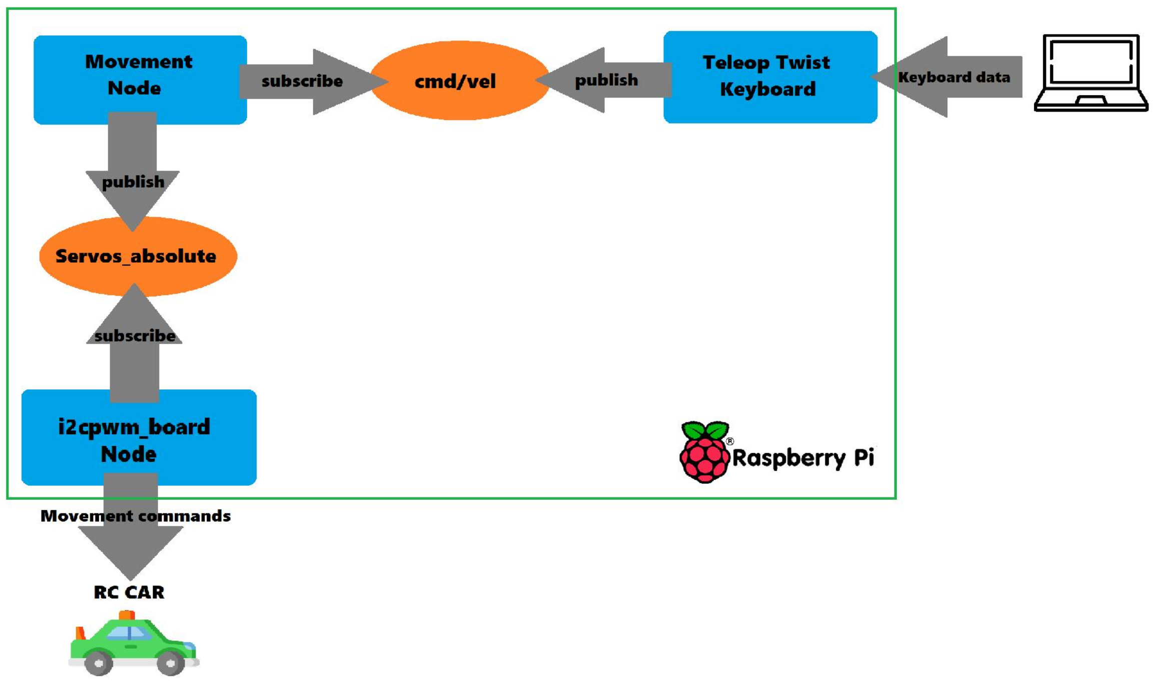


Figure 3.4. ROS architecture for manual control.

In our project, a powerful package Teleop Twist Keyboard is used which comes preinstalled with ROS kinetic. To run this node we just execute the following command:

```
rosrunteleop_twist_keyboard teleop_twist_keyboard.py
```

The movement node will be responsible to convert keyboard command that subscribe from cmd/vel topic and publish it to servo_absolute topic as low level.

I2cpwm_board will generate PWM signals to control the servo and dc motor this node will subscribe from servo_absolute topic and gives movement command to the RC Car.

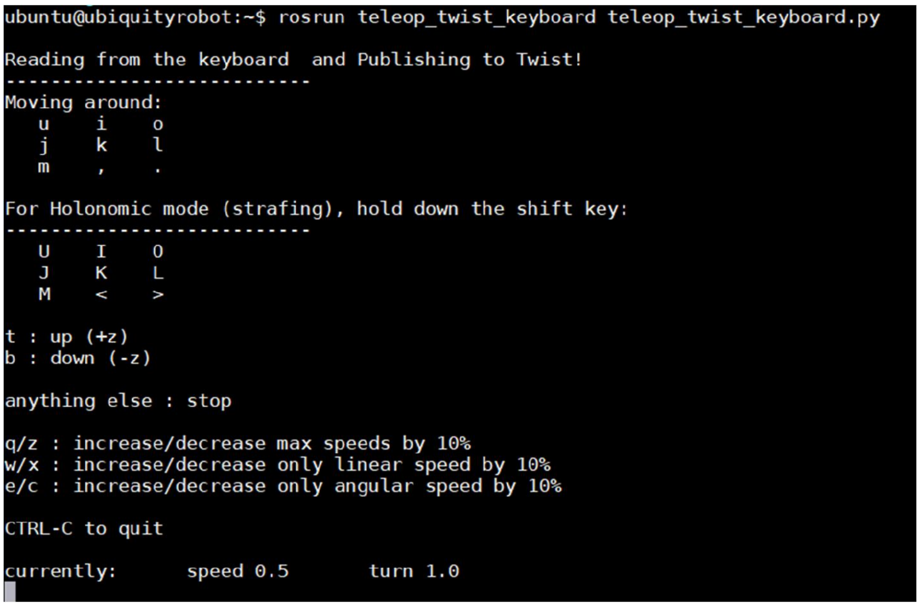
Create launch file

We will create launch file that containsMovement node and i2cpwm_board node so we can launch the two node in one terminal using the command:

roslaunchmovment.launch

The launch file is created using XML and it is given in Appenddix A.

Figure 3.5shows the ROS node teleop_twist_keyboard running in the terminal where the user has the ability to move the car around and increase and decrease the speed of the car.



```
ubuntu@ubiquityrobot:~$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

Figure 3.5. Teleop Twist Keyboard node running in the terminal.

III.3.2.Obstacle detection

In this section we will interface the three ultrasonic sensors with ROS and implement the obstacle avoidance algorithm. Figure 3.6 shows the ROS architecturefor obstacle detection.

The Sonar scan node will scan the three ultrasonic sensors and publish it into three topics for each sensor Right, center and left. The sensor interface node will subscribe to those three topics.

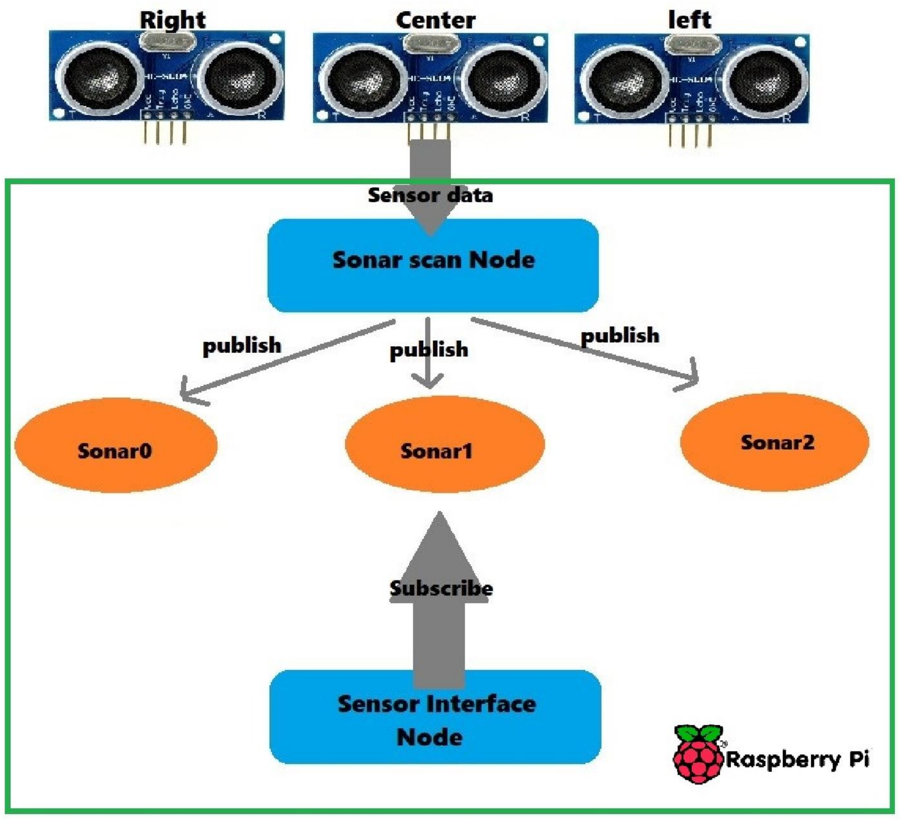


Figure 3.6. ROS architecture for Obstacle detection.

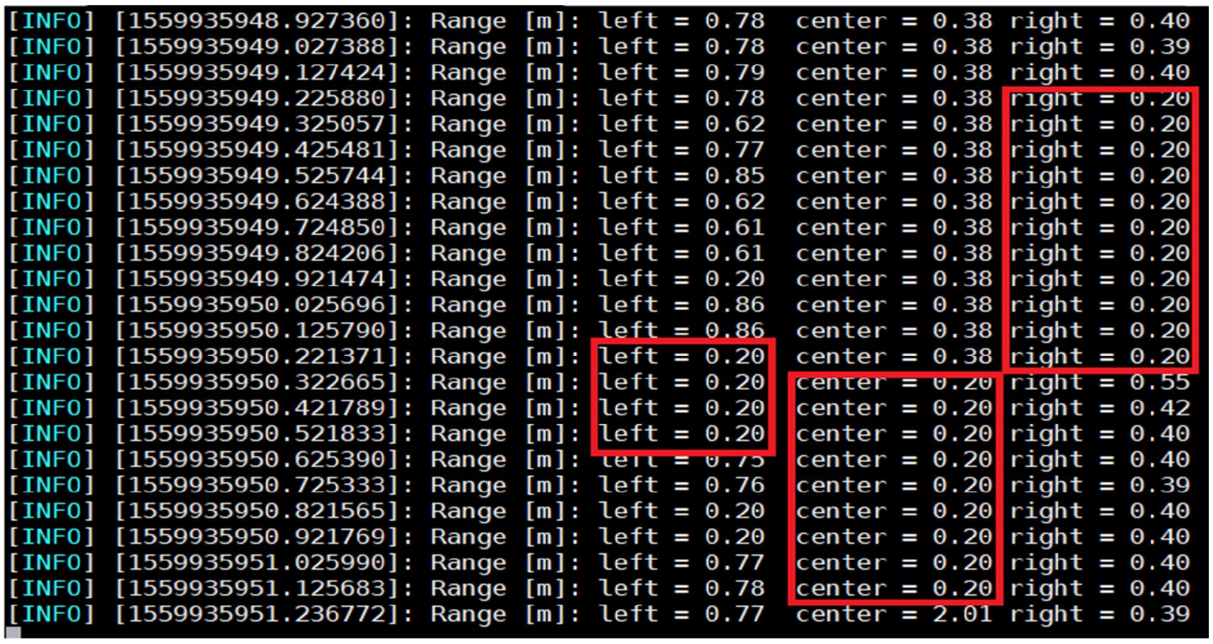
Create launch file

We will create launch file that contains Sensor scan node and sensor interface node so we can launch the two nodes in one terminal using the command:

```
roslaunchsensor_Interface.launch
```

The launch file is created using XML and it is given in Appenddix A.

Figure 3.7 shows how the sensor_interface_node prints to the console when an obstacle is near to the ultrasonic sensors.



```
[INFO] [1559935948.927360]: Range [m]: left = 0.78  center = 0.38 right = 0.40
[INFO] [1559935949.027388]: Range [m]: left = 0.78  center = 0.38 right = 0.39
[INFO] [1559935949.127424]: Range [m]: left = 0.79  center = 0.38 right = 0.40
[INFO] [1559935949.225880]: Range [m]: left = 0.78  center = 0.38 right = 0.20
[INFO] [1559935949.325057]: Range [m]: left = 0.62  center = 0.38 right = 0.20
[INFO] [1559935949.425481]: Range [m]: left = 0.77  center = 0.38 right = 0.20
[INFO] [1559935949.525744]: Range [m]: left = 0.85  center = 0.38 right = 0.20
[INFO] [1559935949.624388]: Range [m]: left = 0.62  center = 0.38 right = 0.20
[INFO] [1559935949.724850]: Range [m]: left = 0.61  center = 0.38 right = 0.20
[INFO] [1559935949.824206]: Range [m]: left = 0.61  center = 0.38 right = 0.20
[INFO] [1559935949.921474]: Range [m]: left = 0.20  center = 0.38 right = 0.20
[INFO] [1559935950.025696]: Range [m]: left = 0.86  center = 0.38 right = 0.20
[INFO] [1559935950.125790]: Range [m]: left = 0.86  center = 0.38 right = 0.20
[INFO] [1559935950.221371]: Range [m]: left = 0.20  center = 0.38 right = 0.20
[INFO] [1559935950.322665]: Range [m]: left = 0.20  center = 0.20 right = 0.55
[INFO] [1559935950.421789]: Range [m]: left = 0.20  center = 0.20 right = 0.42
[INFO] [1559935950.521833]: Range [m]: left = 0.20  center = 0.20 right = 0.40
[INFO] [1559935950.625390]: Range [m]: left = 0.75  center = 0.20 right = 0.40
[INFO] [1559935950.725333]: Range [m]: left = 0.76  center = 0.20 right = 0.39
[INFO] [1559935950.821565]: Range [m]: left = 0.20  center = 0.20 right = 0.40
[INFO] [1559935950.921769]: Range [m]: left = 0.20  center = 0.20 right = 0.40
[INFO] [1559935951.025990]: Range [m]: left = 0.77  center = 0.20 right = 0.40
[INFO] [1559935951.125683]: Range [m]: left = 0.78  center = 0.20 right = 0.40
[INFO] [1559935951.236772]: Range [m]: left = 0.77  center = 2.01 right = 0.39
```

Figure 3.7. Sensor Interface node prints to the console when an obstacle is near the ultrasonic sensor.

III.3.3.Obstacle avoidance algorithm

The algorithm of obstacle avoiding were designed by calculating the steering angle to turn the wheels left or right when an obstacle is detected. The RC Car has 3 ultrasonic sensors, the center one is the main one and the side one is used to decide where to steer to. Proportional to how the car is close to the object with the parameter k_steer the steer angle is calculated using the formula eq.3.1 (as illustrated in figure 3.8):

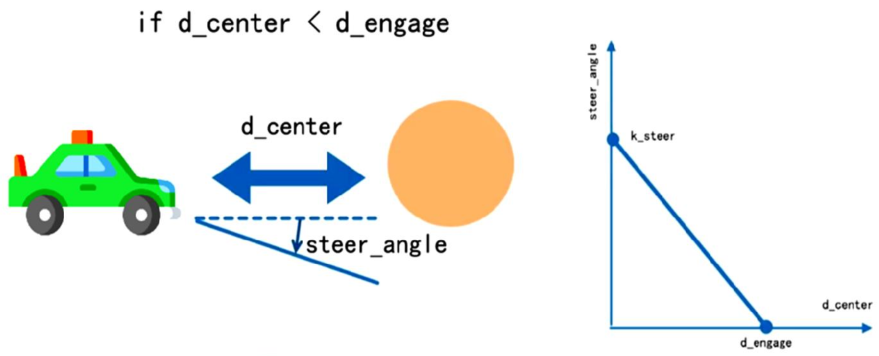
$$\text{Steer\_angle} = \text{k\_steer}\left(1 - \text{d\_center}/\text{d\_angage}\right) \qquad (3.1)$$

Where: Steer_angle: is the steer angle;

K_steer: is the parameter indicating how the car is close to the object.

d_center: it is the distance captured by the centre ultrasound sensor.

d_angage: it is a constant equal to 1.2m.



$$\boxed{steer\_angle = k\_steer\left(1 - \frac{d\_center}{d\_engage}\right)}$$

Figure 3.8.Calculation of the steering angle.

The direction is decided by the side ultrasonic sensors; if left obstacle is detected the car turn right like shown in figure 3.9.

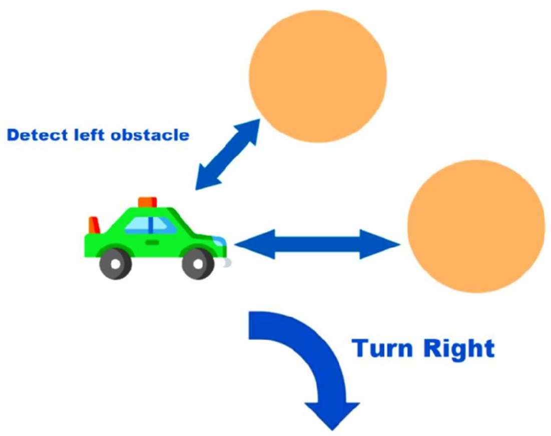


Figure 3.9.Example of how the car detect obstacle and turn.

III.3.4.Braking algorithm

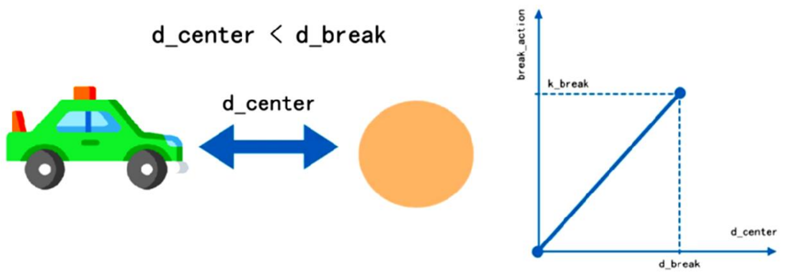
The center ultrasonic sensor is used to control the brakes. This action will be proportional to the distance as shown in figure 3.10 and defined by the equation eq.3.2 below.The value of break will be multiplied by command throttle, value of zero means stop.

$$Break = k\_break(d\_center / d\_break) \quad (3.2)$$

d_center: it is the distance captured by the center ultrasound sensor.

d_break : it is a constant equal to 0.4m.

k_break : it isbraking parameter.



$$\boxed{break = k\_break\left(\frac{d\_center}{d\_break}\right)}$$

Figure 3.10.The braking Algorithm.

After implementing the Obstacle avoidance and braking Algorithms the ROS architecturebecomes as shown in Figure 3.11.

Here after we implement the algorithms with sensor interface node according the values of the three ultrasonic sensors, the sensors interface node will publish to the servos_absolute topic the next move of the car, the i2cpwm_board node will subscribe the data from servos_absolute topic and generate pwm to move the car.

Create launch file

We will create launch file that contains Sensor scan node and sensor interface node and i2cpwm_board node; so, we can launch the three nodes in one terminal using the command:

```
roslaunchobstacleavoidance.launch
```

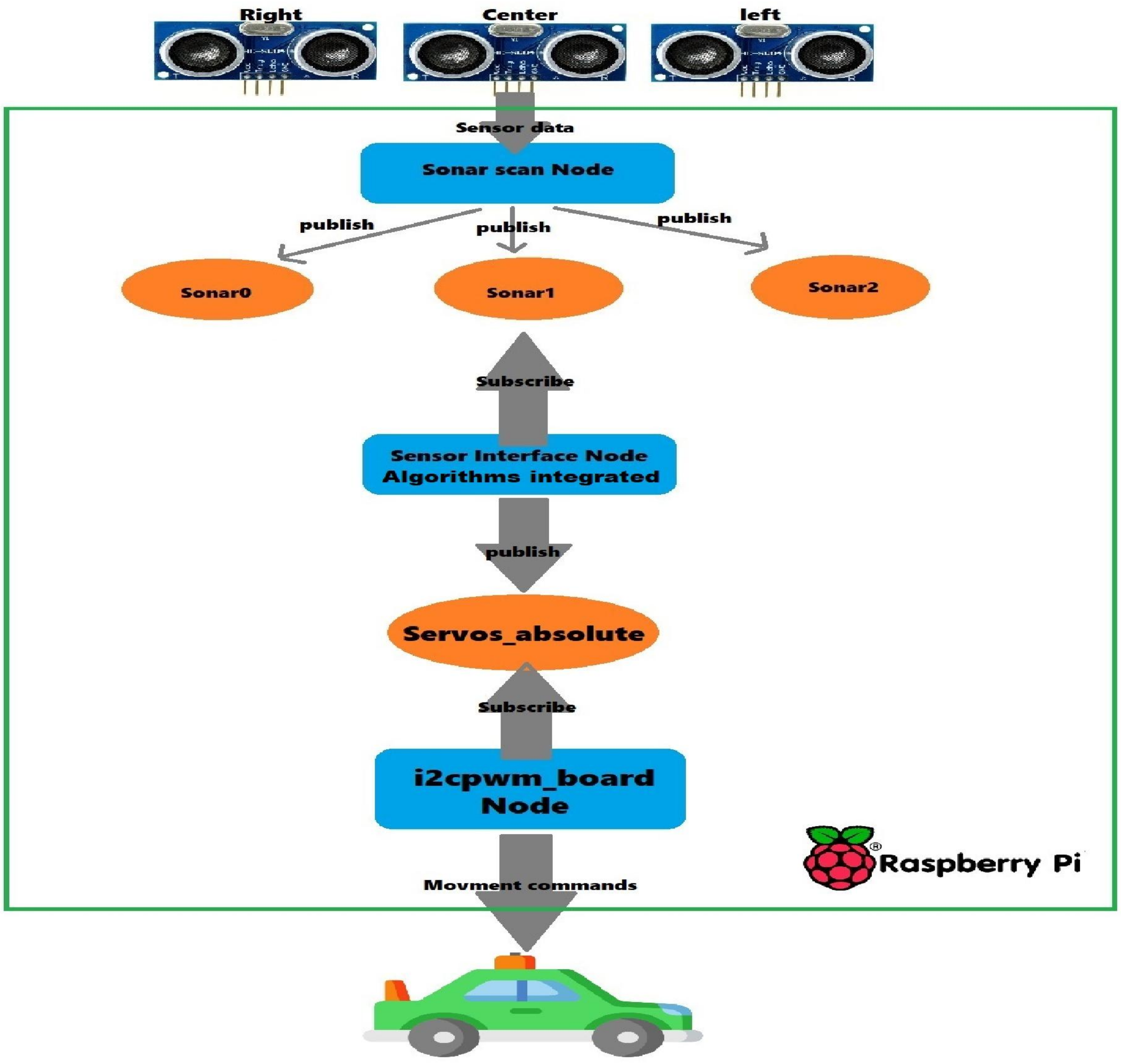


Figure 3.11. ROS architecture for Obstacle avoidance and braking.

III.4.InterfacingPi camera with ROS

The goal of this sectionis to interface Pi camera with ROS. The idea is that, in future work, this function will be used to detect traffic light colours.

In Figure 3.12, the Pi camera streams a video to camera_talker node. Whereas, figure 3.12 illustrates the ROS architecture for Pi camera.

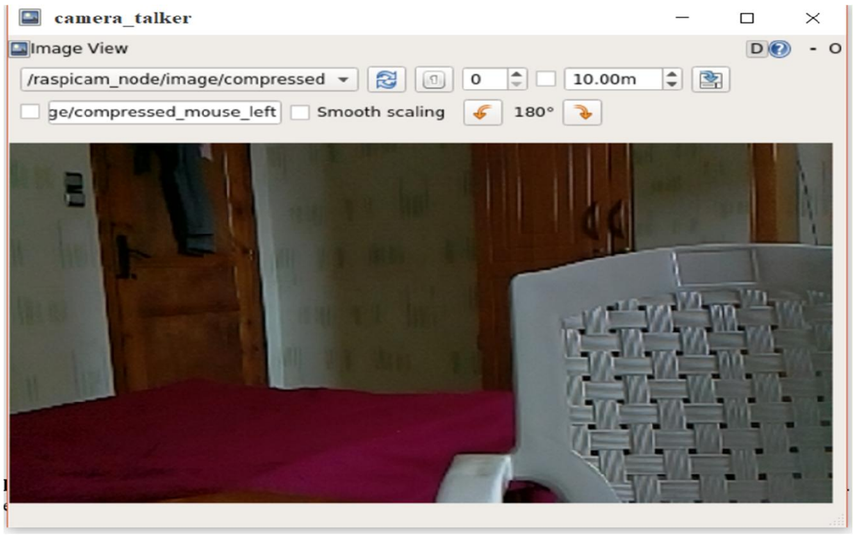
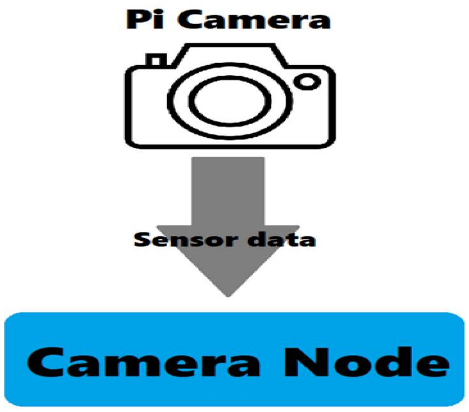


Figure 3.12. Screenshot from the video obtained by the ROS node camera_talker.

Figure 3.13. ROS architecture for Pi camera.



III.5. Interfacing IMU with ROS

The goal of the fourth iteration is to interfaceIMUwith ROS. It will be different then the other sensors because IMU chip is connected with Arduino and not connected directly to the raspberry pi; so, we need to establish serialcommunication between Arduino and ROS (Raspberry Pi) as shown in figure 3.14. For this purpose the ros package **rosserial** is used which facilitates for us to send IMU data to tinyImu topic.

The instructions given in Appendix B show the installation and setting up Arduino with ROS kinetic (Raspberry Pi 3) using Rosserial.

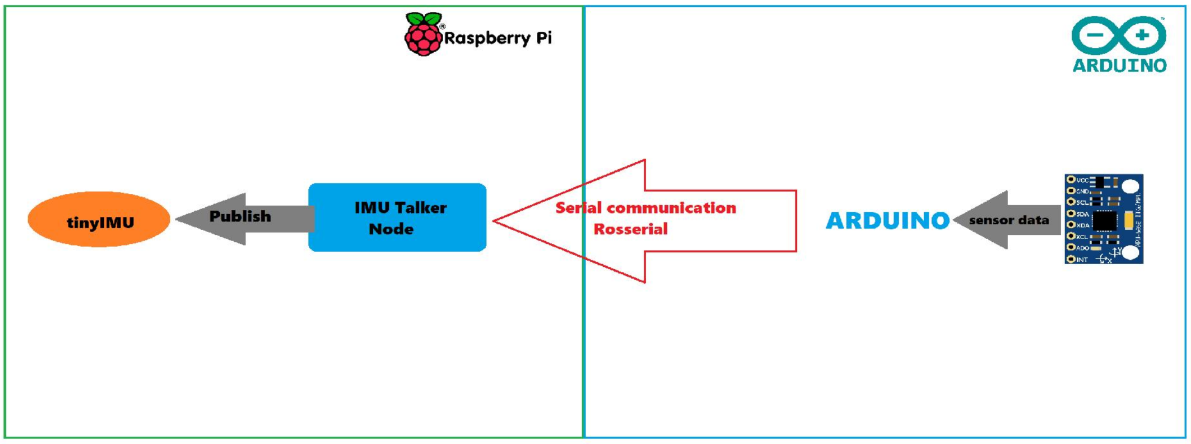


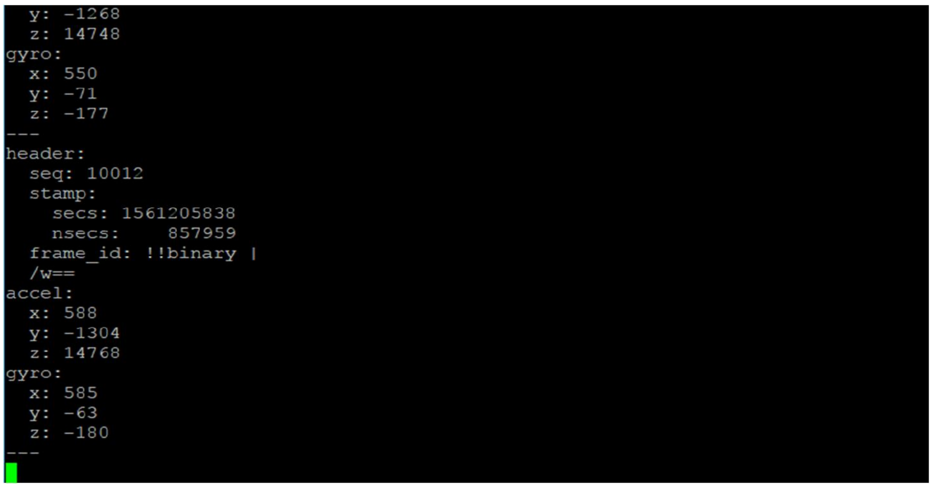
Figure 3.14. ROS architecture for IMU.

The Arduino will read IMU data, after that we start the serial communication using the command:

```
rosrunrosserial_python serial_node.py /dev/ttyUSB0
```

ttyUSB0 is the serial port for Arduino.

The IMU talker node will receive the IMU data and publish it to tinyImu topic, by using the command `rostopic echo / tinyIMU`; we can see the IMU data printed in the consol which means the interfacing is done successfully (IMU data results are shown in figure 3.15).



```
  y: -1268
  z: 14748
gyro:
  x: 550
  y: -71
  z: -177
---
header:
  seq: 10012
  stamp:
    secs: 1561205838
    nsecs:     857959
  frame_id: !!binary |
  /w==
accel:
  x: 588
  y: -1304
  z: 14768
gyro:
  x: 585
  y: -63
  z: -180
---
```

Figure 3.15.tinyIMU topic prints the IMU data.

III.6.ROS Architecture for the whole system

After we done with interfacing all sensors with ROS and implementing some self driving algorithms; now, we connect all the system together. The overall system architecture is given in figure 3.16.We create launch file to launch all the node together in one single command and this is the powerful of the launch file we don't need to launch each node alone that take us time and it is not efficient when wetest the system.We can launch all the node in one terminal using the commands:**roslaunchrccar.launch**

III.7.Route planning

The Route planning is very important in the development of self driving car but we didn't implement it in our RC Car because the lack of LIDAR to map the environment.

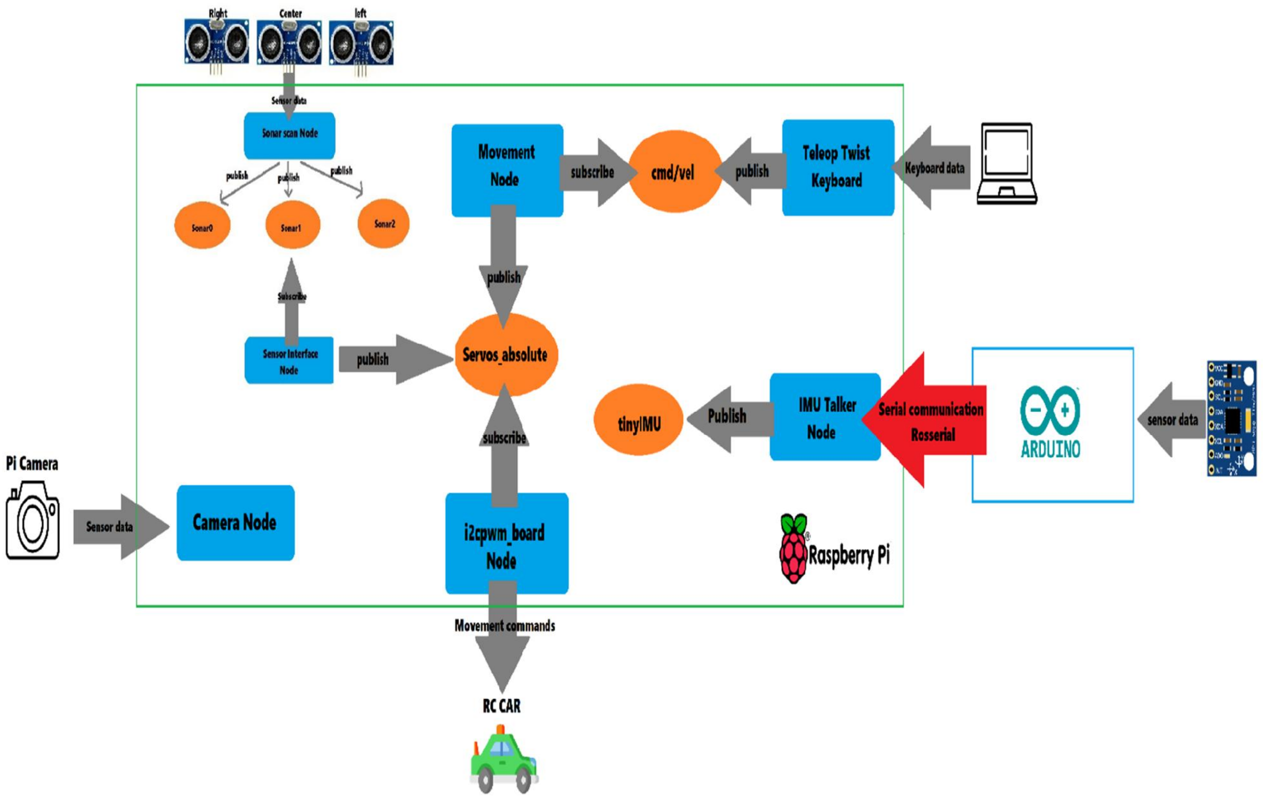


Figure 3.16.ROS Architecture for the whole system.

III.8.Conclusion

In this chapter, we have seen the ROS implementation and how our software is developed step by step.Moreover, the ROS architecture for our RC Car was described and the way the system works was also explained.This chapter dealt with the software implementation andserial communication using Rosserial.

In this chapter we discuss the simulation of the self driving car scheme, the difficulties, the outcomes and the suggestions we have created.

The simulation software does not require strict error management, because the entries are accurately known and the workplace is error free. However, certain mistakes may happen when applying the robot software. The sensors can't answer, the camera may not generate a file and many mistakes can happen.

In this chapter, however, we will discuss simulation which we used in the project with Gazebo because the Self Driving Car problem design method requires a simulator to evaluate and validate our solution under different test situations, and to identify obstacles surrounding the car. In addition, for some other functions like tracking, obstruction prevention, etc…, graphical display may be used.

First, we will discuss a self-driving car's fundamental ideas. In this chapter, some of the sensors used in vehicles are simulated. Here is the list of devices we will simulate and interface with ROS:

1. Laser scanner
2. Camera
3. GPS
4. IMU

We will discuss how to set up the simulation using ROS and Gazebo, the, read the sensor values. This sensor interfacing will be useful when we build our own self-driving car simulation for the first time.

4.1. Simulation

4.1.1. Simulating a laser scanner

This subsection deals with the simulation of a Gazebo laser scanner. It is simulated by offering our applications with custom parameters. If we add ROS, several default Gazebo plugins including the Gazebo laser scanner plug-in can also be installed automatically.

This plugin is used with our customized parameters. So, in essence, we duplicate and construct the set into the workspace using the catkin make command. The basic simulation of the laser scanner, camera, IMU, ultrasonic sensor and GPS is provided in this set.

First we should install a package called hector-gazeboplugins using the command below. This package contains Gazebo plugins of several sensors that can be used in self-driving car simulations.

$ sudo apt-get install ros-kinetic-hector-gazebo-plugins

We just use the following command:

$ roslaunch sensor_sim_gazebo laser.launch

The performance of the laser scanner is first examined. Then, the laser data in Rviz can be visualized, as shown in Figure 4.1 with the command of the rosrun rviz rviz. The topic of the laser information is /laser / scan. For viewing this information, we connect a LaserScan monitor:

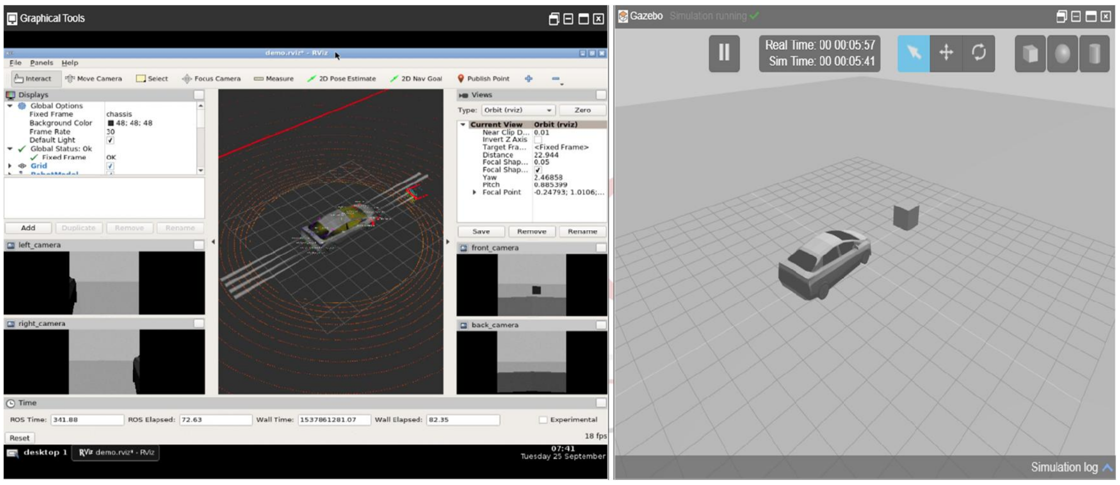


Figure 4.1. Visualization of laser scanner data in Rviz.

When we launch the preceding command, we see an empty world with bmw car made by gazebo as shown in figure 4.1.

4.1.2. Simulating camera in Gazebo

For all sorts of robots, camera is a significant detector. We are going to see how we simulate the starter. To begin the simulations and lunch the camera node, the following command can be used: **$ rosrun rqt_image_view rqt_image_view**

Then, the image can be viewed from the camera using Rviz as illustrated in figure 4.2.

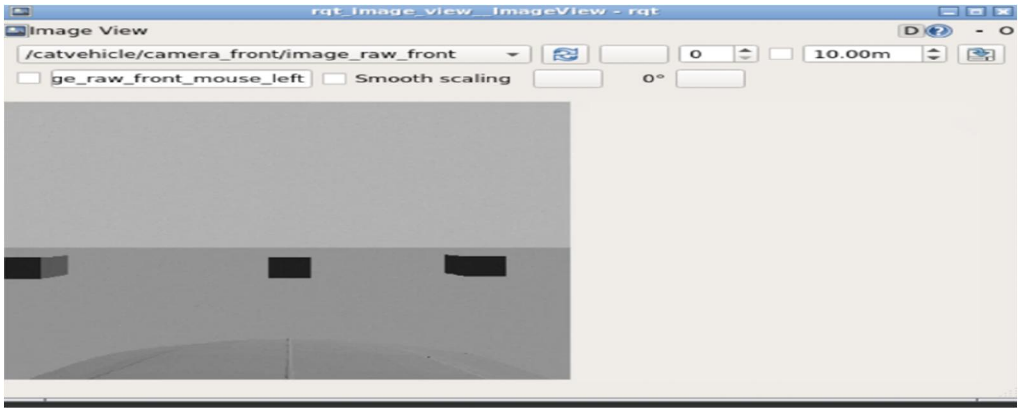

rqt_image_view__ImageView - rqt
Image View
/catvehicle/camera_front/image_raw_front
Smooth scaling
10.00m

Figure 4.2. Image from simulated camera.

To view images from a simulated stereo camera, the following commands are used:

```
$ rosrun image_view image_view image:=/stereo/camera/right/image_raw
$ rosrun image_view image_view image:=/stereo/camera/left/image_raw

$ rosrun image_view image_view image:=/stereo/camera/backright/image_raw
$ rosrun image_view image_view image:=/stereo/camera/backleft/image_raw
```

These commands will display four image windows from each camera of the stereo camera, which are shown in figure 4.3.

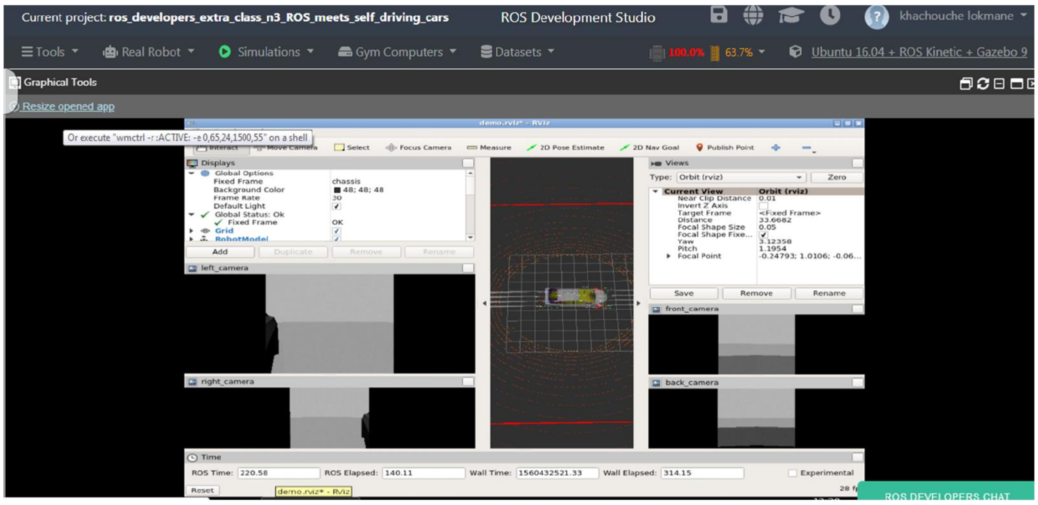


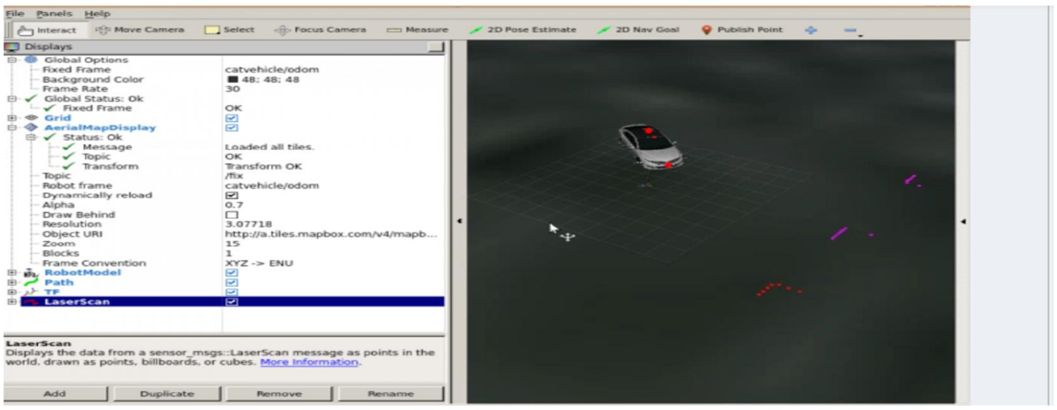
Figure 4.3. Image from simulated stereo camera.

4.1.3. Simulating GPS in Gazebo

GPS is one of the essential sensors in a self-driving car. We can start a GPS simulation using the following command:

$ roslaunch sensor_sim_gazebo gps.launch

We edit rviz config documents from the system, we download and source the rviz satellite set in order to prevent problems with the consent to rviz if we operate rviz in the shell. Therefore, we create a map similar to the simulated world and a globe create a precise copy of it and set the coordinates appropriately (see figure 4.4).



```
Global Options
  Fixed Frame             catvehicle/odom
  Background Color        ■ 48; 48; 48
  Frame Rate              30
✓ Global Status: Ok
  ✓ Fixed Frame           OK
Grid                      ✓
AerialMapDisplay          ✓
  ✓ Status: Ok
    ✓ Message             Loaded all tiles.
    ✓ Topic               OK
    ✓ Transform           Transform OK
  Topic                   /fix
  Robot frame             catvehicle/odom
  Dynamically reload      ✓
  Alpha                   0.7
  Draw Behind             ☐
  Resolution              3.07718
  Object URI              http://a.tiles.mapbox.com/v4/mapb...
  Zoom                    15
  Blocks                  1
  Frame Convention        XYZ -> ENU
RobotModel                ✓
Path                      ✓
TF                        ✓
LaserScan                 ✓

LaserScan
Displays the data from a sensor_msgs::LaserScan message as points in the
world, drawn as points, billboards, or cubes. More Information.
```

Figure 4.4. Visualization of gps data in Rviz

4.1.4. Simulating IMU on Gazebo

We start the IMU simulation using the following command:

$ roslaunch sensor_sim_gazebo imu.launch

We get the orientation values, linear acceleration, and angular velocity from this plugin like we have done in chapter 3.

We have also visualized IMU data in Rviz or in the simulator mode by choosing the simulator mode as illustrated in figure 4.5.

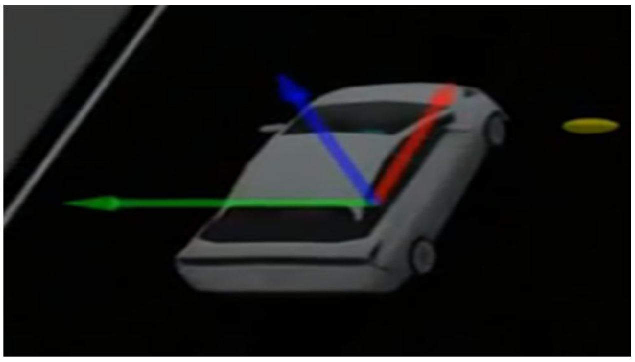


Figure 4.5. Visualization of the /imu in the simulation mode

4.2. Discussion of the SW, HW implementation and ROS platform

In this section we will discuss the HW and SW implementation and benefits also challenges of using ROS to build a self driving car. Because the lack of HW, we didn't acquired all the sensors like LIDAR, GPS, RADAR, Wheels encoder but the work achievement was very satisfying. Figure 4.6 is a photograph of our RC car robot with all the required equipment mounted on it.

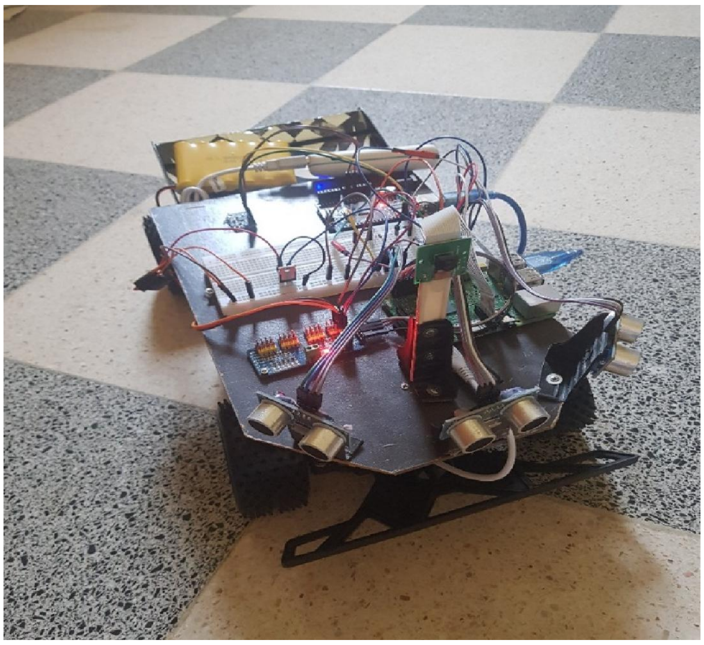


Figure 4.6. Our RC Car in Action

4.2.1. ROS platform analysis

After working with ROS in this project, we can say that it is easy to use and very simple and helpful because it provide us all the tools we need to build robotic project for free. The most important thing is that we do not need to write the device drivers and communications framework, it is all provided by ROS platform.

4.2.2. Benefits of using ROS

The most important thing in ROS that, it is an open source which allow us to add thing and see others implementation; also, it is free and provide us many tools for simulation and visualization of our sensor data using Rviz. It is multi-language and we can create our nodes using either c++ or python.

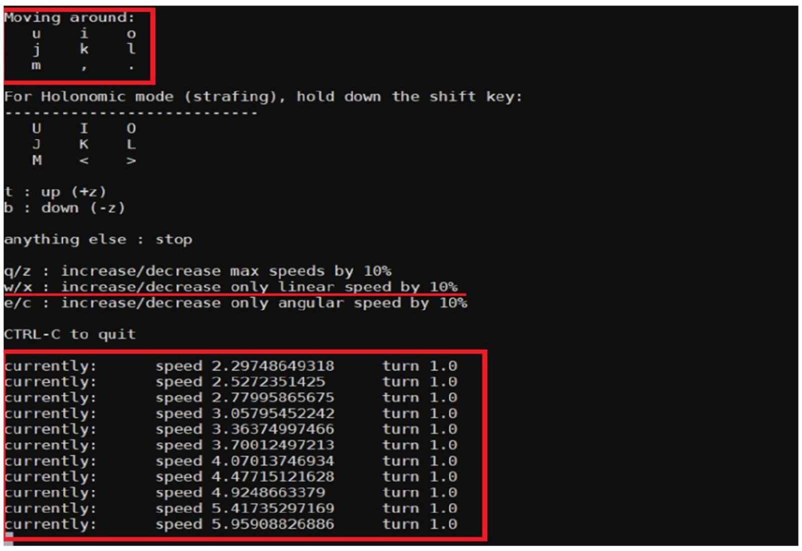
4.2.3. Challenges

The challenges that can face us when we work with ROS are: that we need solid programming language skills (c++ or python) and a basic Linux knowledge because the operating system is ubuntu. We face other problems also like installing the ROS kinetic which was very hard and with a lot of errors so we manage this by using ubiquity robotics version that come pre installed with ROS.

4.2.4. Results

This section presents the Results of SW implementation with HW :

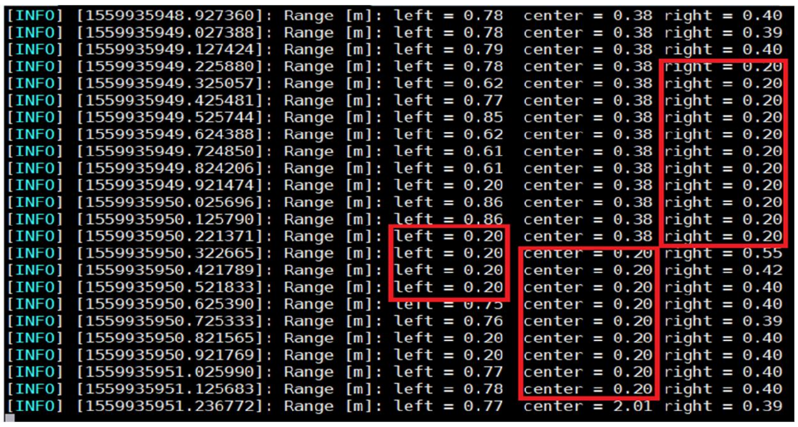
- As we can see in Figure 4.7, we could move the car around with decreasing and increasing the speed. To test it in our RC Car we needed to connect to Raspberry pi via ssh and open two terminal. In the first terminal we launched the teleop_twist_keyboard node and in the second terminal we launched the movement.launch. After following correctly these steps, the RC Car manual control worked perfectly.



```
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 2.29748649318     turn 1.0
currently:      speed 2.5272351425      turn 1.0
currently:      speed 2.77995865675     turn 1.0
currently:      speed 3.05795452242     turn 1.0
currently:      speed 3.36374997466     turn 1.0
currently:      speed 3.70012497213     turn 1.0
currently:      speed 4.07013746934     turn 1.0
currently:      speed 4.47715121628     turn 1.0
currently:      speed 4.9248663379      turn 1.0
currently:      speed 5.41735297169     turn 1.0
currently:      speed 5.95908826886     turn 1.0
```

Figure 4.7. Teleop Twist Keyboard node

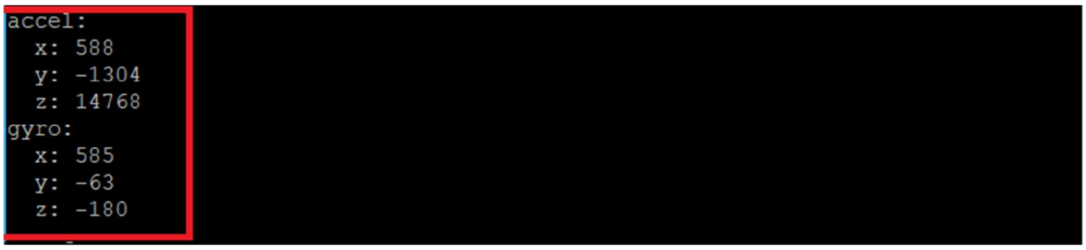
- In Obstacle detection, when we close an object to the three ultrasonic sensors, each one detects the distance to the object in meter and 0.2 m is the minimum value like it is shown in figure 4.8, which means the object is in the danger area.



```
[INFO] [1559935948.927360]: Range [m]: left = 0.78  center = 0.38 right = 0.40
[INFO] [1559935949.027388]: Range [m]: left = 0.78  center = 0.38 right = 0.39
[INFO] [1559935949.127424]: Range [m]: left = 0.79  center = 0.38 right = 0.40
[INFO] [1559935949.225880]: Range [m]: left = 0.78  center = 0.38 right = 0.20
[INFO] [1559935949.325057]: Range [m]: left = 0.62  center = 0.38 right = 0.20
[INFO] [1559935949.425481]: Range [m]: left = 0.77  center = 0.38 right = 0.20
[INFO] [1559935949.525744]: Range [m]: left = 0.85  center = 0.38 right = 0.20
[INFO] [1559935949.624388]: Range [m]: left = 0.62  center = 0.38 right = 0.20
[INFO] [1559935949.724850]: Range [m]: left = 0.61  center = 0.38 right = 0.20
[INFO] [1559935949.824206]: Range [m]: left = 0.61  center = 0.38 right = 0.20
[INFO] [1559935949.921474]: Range [m]: left = 0.20  center = 0.38 right = 0.20
[INFO] [1559935950.025696]: Range [m]: left = 0.86  center = 0.38 right = 0.20
[INFO] [1559935950.125790]: Range [m]: left = 0.86  center = 0.38 right = 0.20
[INFO] [1559935950.221371]: Range [m]: left = 0.20  center = 0.38 right = 0.20
[INFO] [1559935950.322665]: Range [m]: left = 0.20  center = 0.20 right = 0.55
[INFO] [1559935950.421789]: Range [m]: left = 0.20  center = 0.20 right = 0.42
[INFO] [1559935950.521833]: Range [m]: left = 0.20  center = 0.20 right = 0.40
[INFO] [1559935950.625390]: Range [m]: left = 0.75  center = 0.20 right = 0.40
[INFO] [1559935950.725333]: Range [m]: left = 0.76  center = 0.20 right = 0.39
[INFO] [1559935950.821565]: Range [m]: left = 0.20  center = 0.20 right = 0.40
[INFO] [1559935950.921769]: Range [m]: left = 0.20  center = 0.20 right = 0.40
[INFO] [1559935951.025990]: Range [m]: left = 0.77  center = 0.20 right = 0.40
[INFO] [1559935951.125683]: Range [m]: left = 0.78  center = 0.20 right = 0.40
[INFO] [1559935951.236772]: Range [m]: left = 0.77  center = 2.01 right = 0.39
```

Figure 4.8. Sensor Interface node.

- After this data (Ultrasonic capture distance) is sent to Sensor interface node the obstacle avoidance algorithm will be applied. Hence, the node calculates whether the distance is critical; if it is the case, the car must turn left or right according to the side sensors. If the right one detects an obstacle the car turn left otherwise right. According to the algorithm, if d_engage (set to 1.2m) is greater than d_center which is the distance captured by the center sensor steering angle will be calculated proportional to how the car is close to obstacle with parameter k_steer.
- Also we use the center sensor to control the brakes. If the value of d_center is less then d_brakes, which is equal to 0.4m, the action will be proportional to the distance and it will be multiplied by the command throttle. The value of zero means stop.
- The camera works fine and the camera_talker node gets the video stream from camera; so, the interfacing with ROS is done. In the future step, it is enough to create an image processing algorithm which will be implemented to detect colour.
- The interfacing of Arduino and IMU with ROS is done and data is transferred from arduino side to raspberry side and interfaced with ROS this data can be used to detect car position and other tasks. Figure 4.18 shows the gyroscope data indicating the position of the car and the accelerometer data in 3-axis (x,y,z).



```
accel:
  x: 588
  y: -1304
  z: 14768
gyro:
  x: 585
  y: -63
  z: -180
```

Figure 4.9. tinyIMU topic received IMU data.

4.3. Conclusion

In this chapter, we did the simulation in gazebo simulator of the self driving sensors that we implemented and we didn't implement in HW due to problem in SW or lack of HW components in the website "the constructsim"; where we found a lot of resources that help us to simulate our self driving car. Also we discussed the results of our work, which was good. Unfortunately, due to the DC Motor power and the mechanical constraints, the RC Car cannot move in the ground but when we move up the wheels from the ground they move normally. We changed the RC Car chassis twice but the same problem occurred. Also because of budget limitations we didn't use LIDAR, RADAR and GPS for more autonomous features but we give many suggestions about algorithms to be used to add more autonomous features to the RC Car like Route planning. Nevertheless, the results we have achieved are very satisfying because all self driving car sensors were interfaced with ROS. Moreover, we created for each sensor a node and we connected the entire nodes together with topics and we sent messages from one node to another.

GENERAL CONCLUSION

Autonomous cars are the future of driving; as, we see a lot of company develop cars every day that show us how important this project is in our daily life. It is just the start up and more of feature will come to this project in the future.

ROS development for our RC Car is done step by step as we interfaced all the sensor of self driving car with ROS. Unfortunately, due to lack in HW and time limitation and budget we didn't afford the other sensor like LIDAR, RADAR, GPS, Wheels encoder but the result was good as we achieved a lot of things.

The most important thing in ROS is the tools that are provided such us gazebo to stimulate our robot or to visualize our sensor data using Rviz and all for free; also, it is easy to use it requires just some programming skills and basic Linux knowledge.

By working in this project we represented the ROS platform and the powerful tools like gazebo and Rviz. Moreover, we worked with all ros packages almost from nodes, topics and rosserial to establish serial communication between microcontroller and microprocessor so we can split the tasks between the microprocessor for computation and the microcontroller to gather data from sensors.

The benefits of building a self driving car using ROS was way more than the challenges that show us the power of ROS in robotic applications.

As future work we propose the following points:

1. Include LIDAR, RADAR, Wheels encoders and GPS sensors to perform more autonomous car feature like path planning and mapping the environment.
2. Add traffic light colours detection.
3. Deal with crowded and dynamic environment.
4. Using ROS v2 for more security purpose and tools.

References

[1] Bonnie Gringer, History of autonomous car, Jan 12, 2018, titlemax.

[2] Anderson, James M., et al. *Autonomous Vehicle Technology: A Guide for Policymakers*. RAND Corporation, 2014. *JSTOR.*

[3] Levinson, Askeland, Becker, Dolson, Held, Kammel, S Thrun. (2011). *Towards fully autonomous driving: Systems and algorithms*. 2011 IEEE Intelligent Vehicles Symposium (IV), Baden-Baden, Germany, 163-168. June 5-9, 2011.

[4] J. Levinson, J. Askeland, J. Dolson, and S. Thrun, *Traffic Light Localization and State Detection*, in International Conference on Robotics and Automation, 2011.

[5] Hope Reese, *Updated: Autonomous driving levels 0 to 5: Understanding the differences*, January 20, 2016, TechRepublic.

[6] Discant, Rogozan, Rusu, Bensrhair. (2007). *Sensors for Obstacle Detection - A Survey*. Electronics Technology, 30th International Spring Seminar on, 100-105.

[7] Rajeev Thakur, *Infrared Sensors for Autonomous Vehicles, Recent Development in Optoelectronic Devices*, Ruby Srivastava, IntechOpen, December 20th 2017.

[8] Joseph, L. (2017). ROS Robotics Projects. Birmingham: Packt Publishing Ltd.

[9] Quigley, et al., ROS: an open-source Robot Operating System, ICRA workshop on open source software, vol. 3, no. 3.2, 5, 2009.

[10] Ademovic A., *An introduction to robot operating system: the ultimate robot application framework*, May 10, 2018, Toptal.

[11] Aleksandar Zivkovic, *Development of Autonomous Driving using Robot Operating System*, Master thesis, Madrid, May 2018.

[12] Tellez R., *How to start with self-driving cars using ROS*, May 10, 2018, The Construct.

[13] Network Working Group of the IETF, *The Secure Shell (SSH) Protocol Architecture*, January 2006, RFC 4251.

[14] Gazebosim : http://gazebosim.org/tutorials

[15] Rosserial :https://www.intorobotics.com/installing-and-setting-up-arduino-with-ros-kinetic-raspberry-pi-3/

References

[1] Bonnie Gringer, History of autonomous car, Jan 12, 2018, titlemax.

[2] Anderson, James M., et al. *Autonomous Vehicle Technology: A Guide for Policymakers*. RAND Corporation, 2014. *JSTOR.*

[3] Levinson, Askeland, Becker, Dolson, Held, Kammel, S Thrun. (2011). *Towards fully autonomous driving: Systems and algorithms*. 2011 IEEE Intelligent Vehicles Symposium (IV), Baden-Baden, Germany, 163-168. June 5-9, 2011.

[4] J. Levinson, J. Askeland, J. Dolson, and S. Thrun, *Traffic Light Localization and State Detection*, in International Conference on Robotics and Automation, 2011.

[5] Hope Reese, *Updated: Autonomous driving levels 0 to 5: Understanding the differences*, January 20, 2016, TechRepublic.

[6] Discant, Rogozan, Rusu, Bensrhair. (2007). *Sensors for Obstacle Detection - A Survey*. Electronics Technology, 30th International Spring Seminar on, 100-105.

[7] Rajeev Thakur, *Infrared Sensors for Autonomous Vehicles, Recent Development in Optoelectronic Devices*, Ruby Srivastava, IntechOpen, December 20th 2017.

[8] Joseph, L. (2017). ROS Robotics Projects. Birmingham: Packt Publishing Ltd.

[9] Quigley, et al., ROS: an open-source Robot Operating System, ICRA workshop on open source software, vol. 3, no. 3.2, 5, 2009.

[10] Ademovic A., *An introduction to robot operating system: the ultimate robot application framework*, May 10, 2018, Toptal.

[11] Aleksandar Zivkovic, *Development of Autonomous Driving using Robot Operating System*, Master thesis, Madrid, May 2018.

[12] Tellez R., *How to start with self-driving cars using ROS*, May 10, 2018, The Construct.

[13] Network Working Group of the IETF, *The Secure Shell (SSH) Protocol Architecture*, January 2006, RFC 4251.

[14] Gazebosim : http://gazebosim.org/tutorials

[15] Rosserial :https://www.intorobotics.com/installing-and-setting-up-arduino-with-ros-kinetic-raspberry-pi-3/

APPENDIX A

1. Launch files

1.1. Manual control

```
<launch>

             <include file="$(find i2cpwm_board)/launch/i2cpwm_node.launch"/>
             <node pkg="interface_rccar" name="movement" type="low_level_control.py"
           output="screen" >
             </node>
           </launch>
```

1.2. Obstacle detection

```
<launch>
           <node pkg=" interface_rccar " name="sonar_scan" type="sonar_array.py"
           output="screen" >
           </node>
           <node pkg=" interface_rccar " name="sensor_interface" type="
           sensor_interface.py" output="screen" >
           </node>
           </launch>
```

2. Ubuntu install of ROS Kinetic

ROS Kinetic **ONLY** supports Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) for debian packages

1. **Configure our Ubuntu repositories**

Configure our Ubuntu repositories to allow "restricted," "universe," and "multiverse".

2. **Setup our sources.list**

Setup our computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

3. **Set up your keys**

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

4. **Installation**

First, make sure that our Debian package index is up-to-date:

```
sudo apt-get update
```

There are many different libraries and tools in ROS. We provided four default configurations to get us started. We can also install ROS packages individually.

Desktop-Full Install: (Recommended) : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

```
sudo apt-get install ros-kinetic-desktop-full
```

5. **Initialize rosdep**

Before we can use ROS, we will need to initialize rosdep. rosdep enables us to easily install system dependencies for source we want to compile and is required to run some core components in ROS.

```
sudo rosdep init

rosdep update
```

6. **Environment setup**

It's convenient if the ROS environment variables are automatically added to our bash session every time a new shell is launched:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

If we have more than one ROS distribution installed, ~/.bashrc must only source the setup.bash for the version we are currently using.

If we just want to change the environment of our current shell, instead of the above we can type:

```
source /opt/ros/kinetic/setup.bash
```

If we use zsh instead of bash we need to run the following commands to set up our shell:

```
echo "source /opt/ros/kinetic/setup.zsh" >> ~/.zshrc

source ~/.zshrc
```

7. Dependencies for building packages

Up to now we have installed what we need to run the core ROS packages. To create and manage our own ROS workspaces, there are various tools and requirements that are distributed separately. For example, rosinstall is a frequently used command-line tool that enables us to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

APPENDIX B

1. Installing and Setting Up Arduino with ROS Kinetic (Raspberry Pi 3)

This guide will walk us through how to install and setting up an Arduino board to work with Raspberry Pi 3 having in common ROS Kinetic.

To walk through this guide, we must have a Raspberry Pi 3 with ROS Kinetic installed, an Arduino UNO board connected via the USB port to Pi, and some Linux knowledge.

Arduino is an open-source development tool very easy to use both as hardware and software. This development board simplifies the robot construction process and is therefore used together with Raspberry Pi and ROS to control sensors, motors or any other component that can be controlled with a microcontroller.

The Arduino microcontroller can only run one ROS node at a time.

1.1. Install the Arduino IDE

Arduino is connected to Raspberry Pi 3 through the USB port. To program Arduino, we need to install the Arduino IDE on the Pi.

To install the Arduino IDE on the Ubuntu Mate operating system, use the following commands in the Linux terminal.

```
sudo apt-get update

sudo apt-get install arduino arduino-core
```

1.2. Install rosserial

After installing the Arduino IDE, the next step is installing the package that allows communication between ROS and the serial port of the Arduino board. The package is called rosserial_arduino and allows the node that will run on the Arduino to publish or subscribe to the nodes running on Raspberry Pi 3. The rosserial package contains three other packages: rosserial_msgs, rosserial_client, and rosserial_python.

```
sudo apt-get install ros-kinetic-rosserial-arduino

sudo apt-get install ros-kinetic-rosserial
```

After installing the Arduino IDE and the rosserial package, we will first check the IDE, but not before giving Administrator privileges to the current user for the Arduino Permission Checker.

The command is:

```
sudo usermod -a -G dialout your_user_here
```

To open the Arduino IDE, write the following command in the Ubuntu terminal:

```
arduino
```

Once we have checked the installation of the IDE, the next step is to close it and continue the setup operations. To close, use the Ctrl + C keys.

If we give the command *ls* in the Ubuntu terminal, we will find a new sketchbook directory. If we do not want to change us location or name, all Arduino sketches will be saved in this directory.

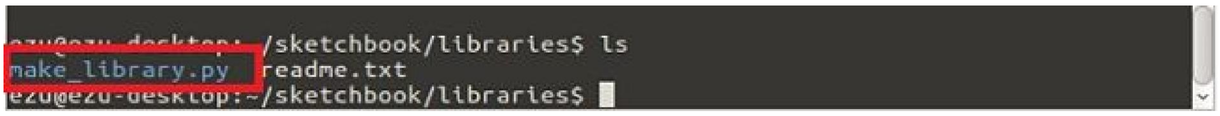
To write Arduino sketches for ROS, we need the ros_lib library.

1.3. Install the ros_lib library

The link between ROS and Arduino is through the ros_lib library. This library will be used as any other Arduino library.

To install the ros_lib library, type the following commands in the Ubuntu terminal:

```
cd sketchbook/libraries

rosrun rosserial_arduino make_library.py .
```

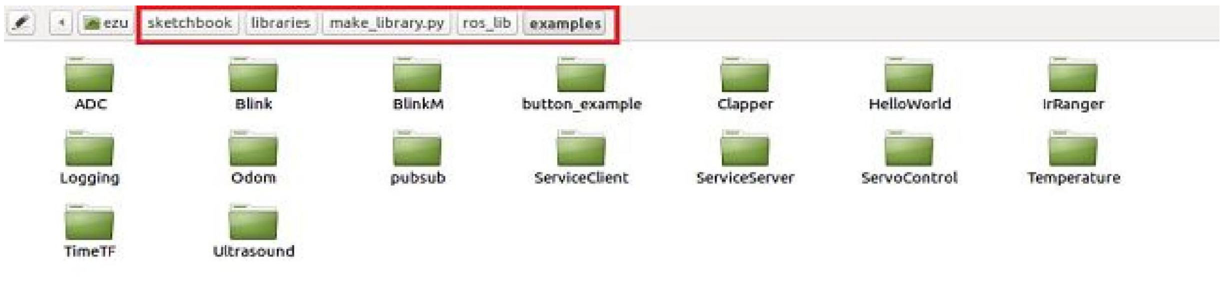


```
ezu@ezu-desktop:~/sketchbook/libraries$ ls
make_library.py  readme.txt
ezu@ezu-desktop:~/sketchbook/libraries$
```

Appendix B.1. Content of Arduino libraries after the commands.

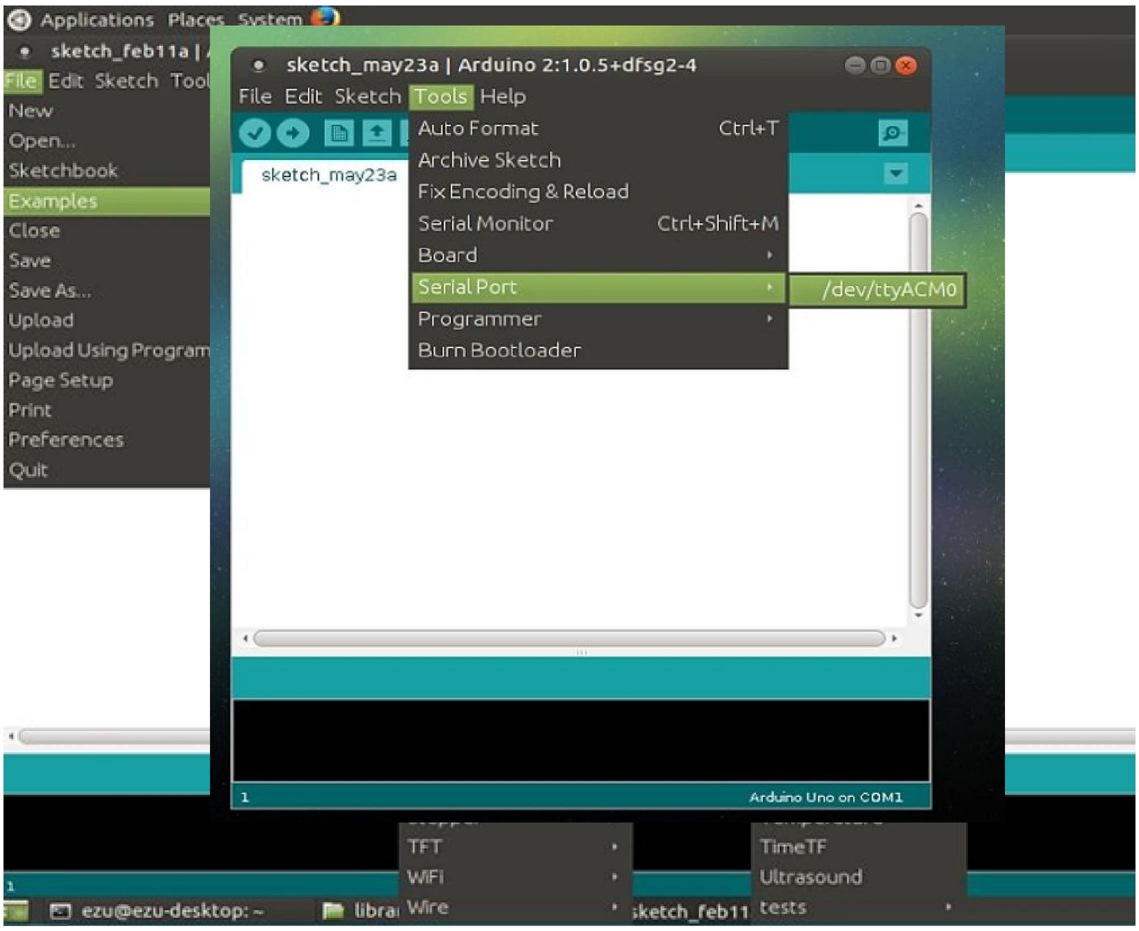
make_library.py

If we browse the **sketchbook/libraries/make_library.py/ros_lib/examples** folder, we will find a list of examples that can be used in ROS projects. One of these examples is Ultrasound. In another tutorial, I will use this example to control one or more HC-SR04 ultrasonic sensors with ROS and Arduino.



Appendix B.2.ros_lib in Example in Arduino IDE

The ros_lib examples



Appendix B.3. connecting serial port.

Before checking the latest IDE settings, we must rename the *'make_library.py'* folder. The Arduino IDE does not allow the use of points in the name of the libraries. So the name of the bookstore will become *'make_library'*.

1.4. Check the Arduino IDE settings

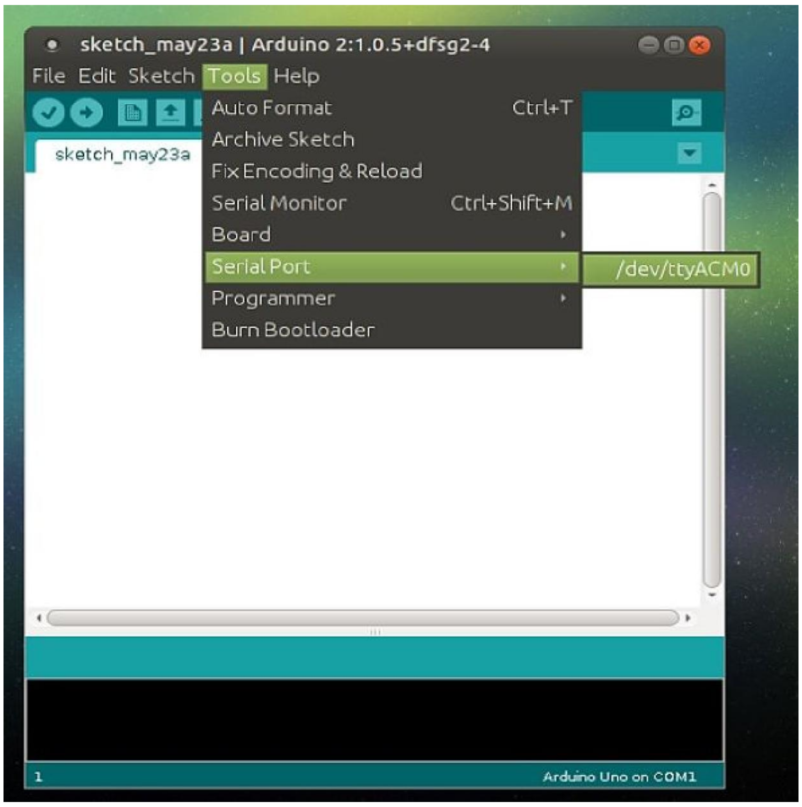
To check the settings made, we will open the Arduino IDE again. To launch the application, we will use the command:

```
arduino
```

After opening the IDE, check if we have access to the ros_lib examples: **File -> Examples -> make_library -> ros_lib**

ros_lib

Check the serial port:
Tools-> Serial Ports



Appendix B.4. Serial port after connecting Arduino with Raspberry Pi

Arduino's serial port

Thus, the steps to install and setup the Arduino IDE on Raspberry Pi 3 with ROS Kinetic are detailed in [15].