

# Table of Contents

<b>Abstract:</b>	<b>2</b>
<b>Chapter 1      <i>The Problem Statement</i></b>	<b>3</b>
Section 1.1 <b>Motivation</b>	<b>3</b>
Section 1.2 <b>Objective Statement</b>	<b>3</b>
Section 1.3 <b>Research Survey</b>	<b>3</b>
1.3.1 Simultaneous localization and mapping (SLAM):	3
1.3.2 Robot Software Platform:	4
1.3.3 Robot Operating System (ROS):	5
1.3.4 Embedded System:	7
1.3.5 RGB-D camera:	7
1.3.6 2D LIDAR scanner:	8
1.3.7 Fisheye camera:	9
1.3.8 Ultrasonic Distance sensor:	9
1.3.9 DC motor:	10
1.3.10 R/C Servo Motor:	10
Section 1.4 <b>User Needs</b>	<b>10</b>
<b>Chapter 2      <i>The Requirement Specification</i></b>	<b>11</b>
Section 2.1 <b>Engineering Requirements</b>	<b>11</b>
Engineering Requirements	11
justification	11
Section 2.2 <b>Full Requirement Specification</b>	<b>11</b>
Engineering Requirements	11
Corresponding Marketing Requirement	11
<b>Chapter 3      <i>The Design</i></b>	<b>13</b>
Section 3.1 <b>Overview of the system</b>	<b>13</b>
3.1.1 Data Collection	13
3.1.2 Processing Unit	15
3.1.3 Movement system	15
3.1.4 ROS environment	18
Section 3.2 <b>Functional Specifications</b>	<b>19</b>
3.2.1 Level 1 design:	20
3.2.2 Workflow diagram:	21
<b>Chapter 4      <i>Implementation and Verification</i></b>	<b>23</b>
Section 1: R/C car chassis	23
Section 2: Software environment	25
<b>Conclusion</b>	<b>28</b>
<b>Appendix A      <i>The Project Plan</i></b>	<b>31</b>
Section 4.1 <b>Work Breakdown Structure</b>	<b>31</b>
Section 4.2 <b>Gantt Chart</b>	<b>32</b>
Section 4.3 <b>Cost Estimation</b>	<b>33</b>
<b>Appendix B      <i>Program Code</i></b>	<b>34</b>
<b>Appendix C      <i>Data sheets</i></b>	<b>50</b>

**Abstract:**

The number of road traffic deaths continues to rise steadily [1], driver distraction is one contributing cause of traffic accidents. This made governments and companies to find a solution to reduce the high number of deaths. One of the solutions is to create Autonomous vehicles (AVs) without human intervention. Therefore, studies have increased in recent years and reached a somewhat satisfactory result.

In this report, we designed a prototype that simulates Autonomous vehicles (AVs) in a smaller environment at R/C car scales. then, we used an RGB-D camera and 2D LIDAR to collect the data from the indoor environment using Simultaneous localization and mapping (SLAM). Raspberry Pi 4B as a computer to run Robot Operating System (ROS) that run applications such as building a map, and object detection, connected to embedded system of an 1:10 scale R/C car.

# Chapter 1      The Problem Statement

## Section 1.1 Motivation

One of the top ten causes of death in low-income and lower-middle countries are the road injury according to World Health Organization (WHO) from 2000 to 2019, the number of road traffic deaths continues to rise steadily, reaching 1.35 million in 2016 around the world [1] involving cars, buses, motorcycles, bicycles, trucks, or pedestrians.

Locally, according to the Najm for Insurance Services in the Kingdom of Saudi Arabia, the company's statistical report estimates the top five main causes of a traffic accident in 2018 [2], are:

- Unsafe lane change.
- Lack of adequate distance between vehicles in the road.
- Driving in reverse.
- Distraction by mobile phones.
- Violating driving priority rules.

The driver distraction is a contributing cause of traffic accidents, leading to raise the number of deaths and economic issues effect on the governments and people.

## Section 1.2 Objective Statement

The continuing evolution of Autonomous Vehicles (AVs) aims to deliver even greater safety benefits than earlier technologies [3]. Instead of building larger scale, building smaller scale achieving some requirements such as Autonomous R/C car.

Autonomous R/C cars could have:

- Less control by human.
- Environment vision or sensing.
- Identify the destination and choose the best path to reach.
- Detect and avoiding the objects.
- A suitable power supply achieves mobility feature.

The objective of this project is to design a system and prototyping an autonomous R/C car to simulate outdoor environment for making the transportation much safer by allowing an autonomous system to drive the R/C car, identify and avoid the obstacles, able to use a navigation, and set the destination from current point to the target point.

## Section 1.3 Research Survey

During our research we found these relevant technologies and systems:

### 1.3.1 Simultaneous localization and mapping (SLAM):

A method used for autonomous vehicles that make you build a map and localize autonomous vehicles to detect objects and avoiding them. In addition, using many information such as the number of wheel revolutions to determine the amount of movement and any direction needed, this is called localization. Simultaneously, a

camera or other sensors used to create a map of the obstacles in its surroundings, this is called mapping [4]. Widely applied into mobile robots, self-driving, drones, and autonomous underwater vehicles.

Moreover, divided into two types of algorithms VSLAM and LIDAR SLAM. The traditional visual SLAM systems use stereo camera as input sensor, with complex map initialization and map point triangulation steps needed for 3D map reconstruction, which are easy to fail. The emergence of RGB-D camera which provides RGB image together with depth information breaks this situation [5]. LiDAR(Light detection and ranging) SLAM is a method that primarily uses a laser sensor. The output values from laser sensors are generally 2D (x, y) or 3D (x, y, z) point cloud data. The laser sensor point cloud provides high-precision distance measurements and works very effectively for map construction.

Therefore, SLAM method based on graph-based optimization divides the SLAM problem into two parts: front-end and back-end. The front-end is responsible for the construction of the pose map, and the back-end adjusts the configuration of the graph through optimization. The following data fusion of 2D LiDAR and RGB-D camera for front-end data registration and loop closure detection is used to improve the accuracy of pose image in Figure 1 below [6].

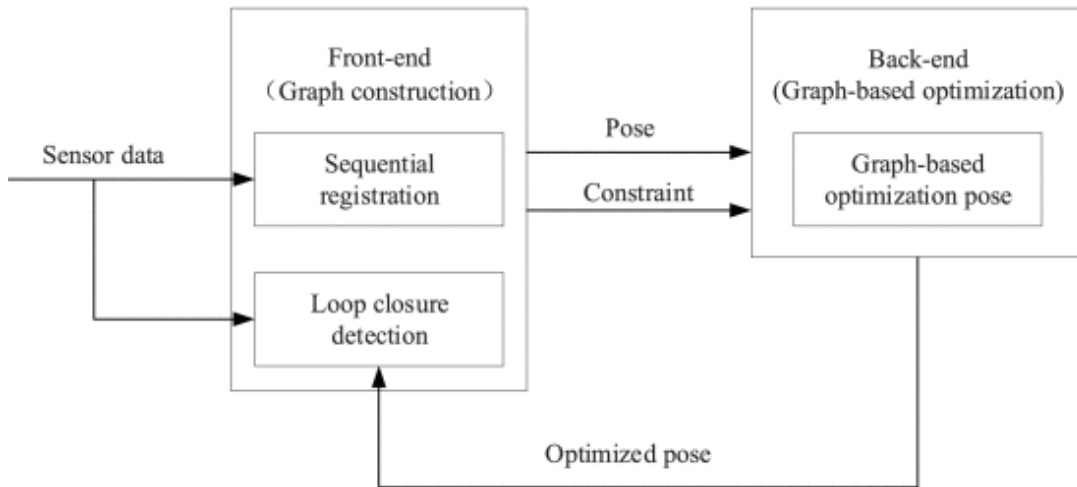


Figure 1: SLAM framework based on graph-based optimization [6].

### 1.3.2 Robot Software Platform:

A Platform divided into Software Platform and Hardware Platform. A robot software platform includes tools that are used to develop robot application programs such as hardware abstraction, low-level device control, sensing, recognition, SLAM (Simultaneous Localization And Mapping), navigation, manipulation and package management, libraries, debugging and development tools. Robot hardware platforms such as mobile robots, drones, and humanoids. Among these software are Robot

Operating System (ROS)<sup>1</sup>, Japanese Open Robotics Technology Middleware (OpenRTM)<sup>2</sup>, European real-time control centered OROCOS<sup>3</sup>, Korean OPRoS<sup>4</sup>, etc. Although, there are too many kinds of robot software causing many complications and problems, the robot researchers from around the world are collaborating to take Robot Operating System (ROS) the most popular robot platform [7].

### 1.3.3 Robot Operating System (ROS):

A set of software libraries and tools that implements the various GUI tools in the form of plugins) and 3D visualization tool (RViz) that can be used without developing necessary tools for robot development [8]. Furthermore, the tool can also receive 3D distance information from recently spotlighted Intel RealSense or Microsoft Kinect and easily convert them into the form of point cloud and display them on the visualization tool. Therefore, ROS supports a variety of operating systems such as Windows (XP, 7, 8, 10), Linux (Linux Mint, Ubuntu, Fedora, Gentoo) and Mac (OS X Mavericks, Yosemite, El Capitan), since Ubuntu is the most widely used among ROS users.

Also, has many concepts you have to know:

- **Master:** acts as a name server for node-to-node connections and message communication. When you execute "roscore", the master will be configured with the URI address (by default uses the IP address of local PC) and 11311 port number.
- **Node:** refers to the smallest unit of processor running to creating one single node for each purpose, For example, the program to operate the robot is divided into specialized functions used for each function such as sensor drive, sensor data conversion, obstacle recognition, motor drive, encoder input, and navigation.

Figure 2 show you how nodes connected with master, the Master provides naming and registration services to the rest of the nodes.

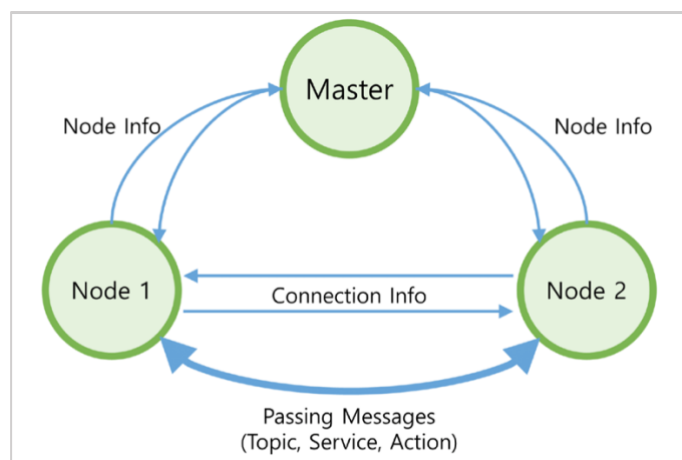


Figure 2: Message Communication.

- **Message:** A node sends or receives data between nodes via a message. Messages are variables such as integer, floating point, and Boolean.
- **Publisher:** The publisher node registers its own information and topic with the master and sends a message to connected subscriber nodes that are interested in the same topic. The publisher is declared in the node and can be declared multiple times in one node.
- **Subscriber:** The subscriber node registers its own information and topic with the master and receives publisher information that publishes relative topic from the master. Based on received publisher information, the subscriber node directly requests connection to the publisher node and receives messages from the connected publisher node.
- **Topic:** The publisher node first registers its topic with the master and then starts publishing messages on a topic. Subscriber nodes that want to receive the topic request information of the publisher node corresponding to the name of the topic registered in the master. Based on this information, the subscriber node directly connects to the publisher node to exchange messages as a topic.
- **Service:** is synchronous bidirectional communication between the service client that requests a service regarding a particular task and the service server that is responsible for responding to requests.
- **Action:** is used where it takes longer time to respond after receiving a request and intermediate responses are required until the result is returned.

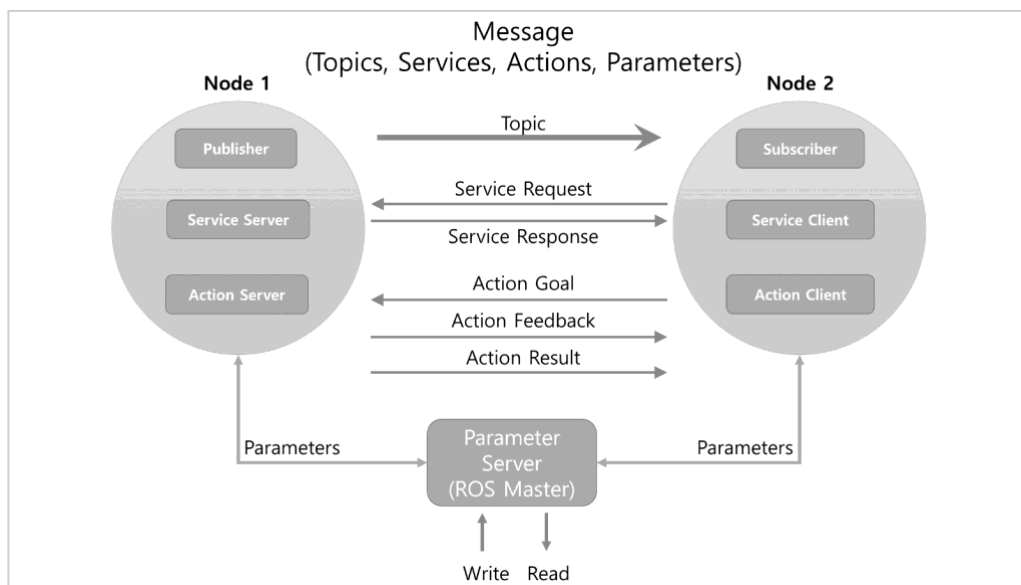


Figure 3: Message Communication between Nodes [7].

In addition, multiple subscribers can receive message from a publisher and vice versa. Moreover, Multiple publishers and subscribers connections are available as well in Figure 4.

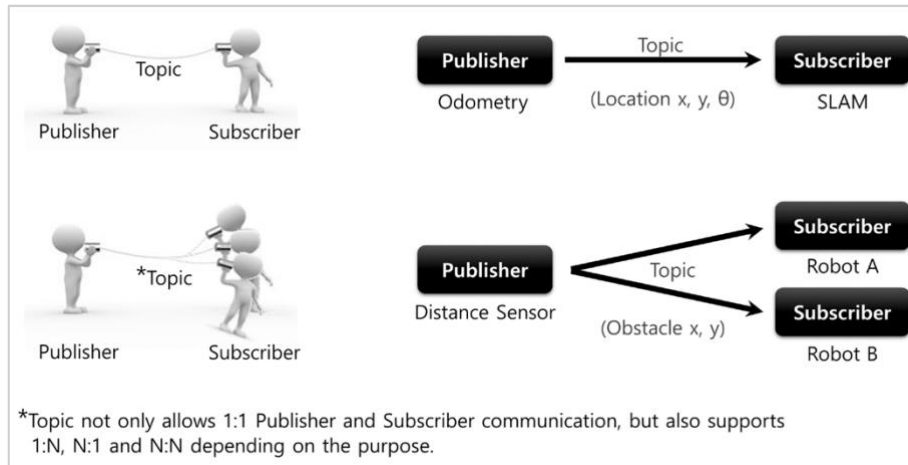


Figure 4: Topic Message Communication [7].

### 1.3.4 Embedded System:

An embedded system is an electronic system that exists within a device as a computer system that performs specific functions for control of a machine or other system that requires control. In other words, an embedded system can be defined as a specific purpose computer system that is a part of the whole device and serves as a brain for systems that need to be controlled.

In particular, a microcontroller capable of real-time control is required to use an actuator or sensor of a robot, and the high-performance processor-based computer is required for image processing using a camera or navigation, manipulation.

Figure 5 shows various systems from 8-bit microcontroller to high-performance PC and you need to configure an embedded system with the right performance to match the needs.


					
	8/16-bit MCU	32-bit MCU		ARM A-class	x86
		"small" 32-bit MCU	"big" 32-bit MCU		
Example Chip	Atmel AVR	ARM Cortex-M0	ARM Cortex-M7	Samsung Exynos	Intel Core i5
Example System	Arduino Leonardo	Arduino M0 Pro	SAM V71	ODROID	Intel NUC
MIPS	10's	100's	100's	1000's	10000's
RAM	1-32 KB	32 KB	384 KB	a few GB (off-chip)	2-16 GB (SODIMM)
Max power	10's of mW	100's of mW	100's of mW	1000's of mW	10000's of mW
Peripherals	UART, USB FS, ...	USB FS	Ethernet, USB HS	Gigabit Ethernet	USB SS, PCIe

Figure 5: Types of embedded boards.

### 1.3.5 RGB-D camera:

Many cameras and sensors are a specific type of depth-sensing devices that work in association with a RGB (red, green and blue color) sensor camera. They are able to augment the conventional image with depth information (related with the distance to

the sensor) in a per-pixel basis. In related information, has many advantages over traditional cameras, like scale awareness and the ability to reconstruct 3D structures for even low texture areas easily and quickly [5]. Moreover, may provide a significant contribution to solve or simplify many challenging tasks, such as object detection, scene parsing, pose estimation, visual tracking, semantic segmentation, shape analysis, image-based rendering, and 3D reconstruction. There are several RGB-D sensors, such as Microsoft Kinect, Structure IO, ASUS Xtion Pro, and Intel RealSense in Figure 6, have appeared on the market [9].



Figure 6: RGB-D cameras.

### 1.3.6 2D LIDAR scanner:

Light detection and ranging sensors provide range data with the high accuracy and resolution in a wide FOV. The system emits laser signal that is reflected by a target and then measured by the LiDAR detector. Due to the known speed of light  $c$ , their measurement of the laser time-of-flight (ToF)  $t$  directly determines the distance by

$$d = \frac{c \times t}{2} [10].$$

Therefore, they perform satisfactorily in bad weather conditions. Otherwise, it cannot provide information about object shape and colour [11]. Furthermore, the system based on lidar is considered as an effective and accurate way for robots to construct a map and locate themselves [12]. See Figure 7.

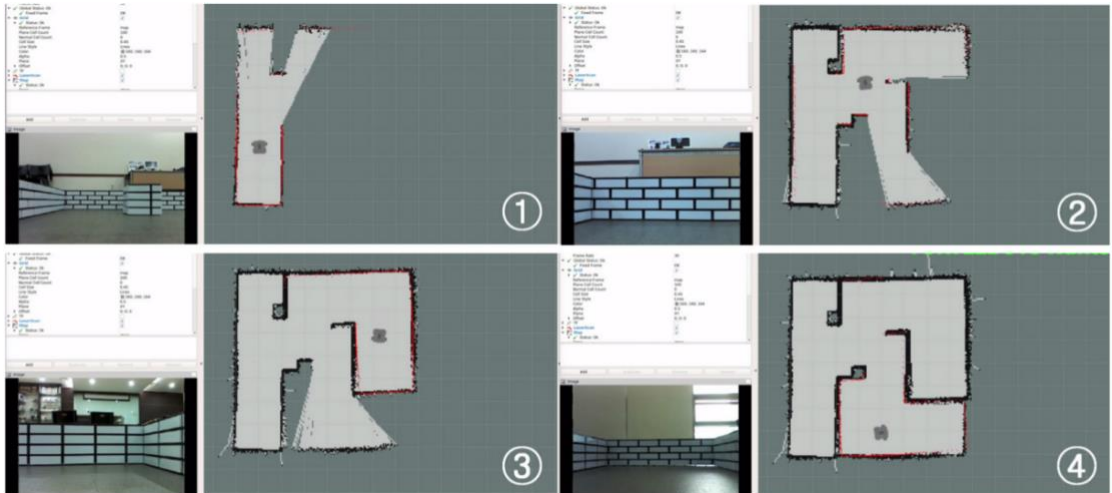


Figure 7: 2D mapping in turtlebot3 [13].



### 1.3.7 Fisheye camera:

Recently fisheye cameras, owing to their large field of view (LFOV), have attracted diverse attention from both technical experts and the public in general. Large FOV cameras are necessary for various computer vision application domains, including video surveillance and augmented reality, and have been of particular interest in autonomous driving [14]. Whereas it comes at the cost of strong radial distortion making a common issue, the image will appear warped in Figure 8 that is not easy for standard computer vision models to deal with it [15].

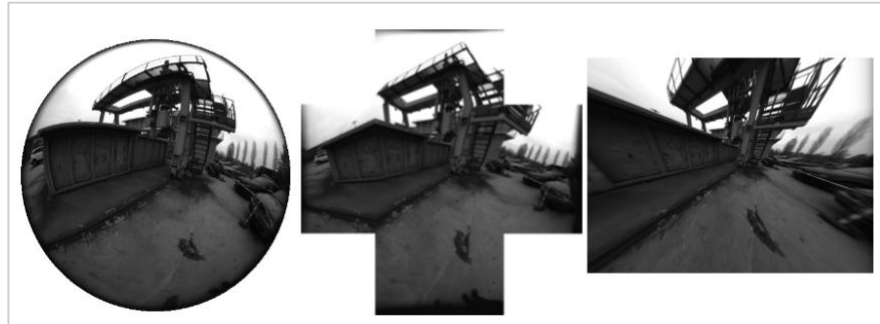


Figure 8: The same image wrapped to fit a three projection models [15].

### 1.3.8 Ultrasonic Distance sensor:

Electronic device that measures the distance of a target object by emitting ultrasonic sound waves and convert the reflected sound into an electrical signal. However, ultrasonic sensors have two main components: the transmitter, emits the sound using piezoelectric crystals. the receiver, encounters the sound after it has travelled to and from the target.

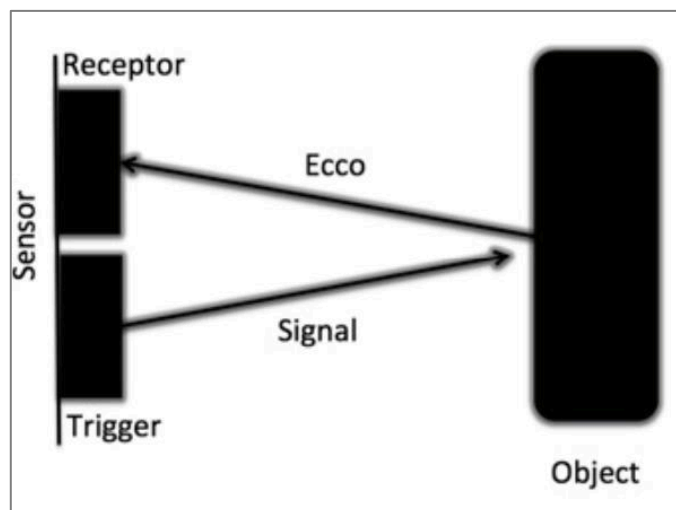


Figure 9: Ultrasonic sensor.

Further, to calculate the distance between the sensor and the object, the sensor measures the time it takes between the emission of the sound by the transmitter to its

contact with the receiver. The formula for this calculation is  $D = \frac{1}{2} T \times C$  (where D is the distance, T is the time, and C is the speed of sound  $\sim 343$  meters/second) [16]. Eventually, multiple ultrasonic sensors are used to increase the accuracy of the obstacles position and decrease the scan time. But, to get the various obstacle information by processing data received from multiple sensors need expensive hardware system. In addition, if the angle of incidence is too large, then the wave reflected from the object does not enter the receiver, which will lead to an incorrect measurement.

#### **1.3.9 DC motor:**

A DC motor or direct current motor is an electrical machine that transforms electrical energy into mechanical energy by creating a magnetic field that is powered by direct current. When a DC motor is powered, a magnetic field is created in its stator. The field attracts and repels magnets on the rotor; this causes the rotor to rotate. To keep the rotor continually rotating, the commutator that is attached to brushes connected to the power source supply current to the motors wire windings. DC motors are able to immediately start, stop, and revers.

#### **1.3.10 R/C Servo Motor:**

An R/C servo is the electric motor, used for controlling the steering wheels of the car. In addition, R/C Servos are used to convert electrical signal into polar or linear movement. When signal is transmitted from the control to the car, this signal is decoded and sent to a servo. According to this signal, the servo will rotate it's drive shaft for some degrees, and this rotation is translated into wheel steering. They are driving them with a PWM circuit [17].

### **Section 1.4 User Needs**

1. The components shall be powered by battery.
2. The system shall be implemented by using Simultaneous Localization and Mapping (SLAM).
3. The system shall be able to detect obstacles and avoid them.
4. The system shall move from an initial point to a target point.
5. The system shall work at indoor environment.
6. The system shall be controlled remotely.

## Chapter 2      The Requirement Specification

### Section 2.1 Engineering Requirements

Engineering Requirements	justification
1. The system shall have RGB-D camera.	The camera collects data and mapping from his environment to identify the objects.
2. The R/C car must use steering system.	To avoid obstacles, we need to steer the wheel away from the nearby objects.
3. The R/C car chassis shall between 1/18 to 1/8 scales.	1/8 scales have 470-595 mm length and 1/18 have 225-230 mm length. The larger cars making it difficult to move in the indoor environment and confined spaces.
4. The components shall operate on batteries for 30 minutes.	Indoor environments have a short distance, which requires a short time and less power to reach the target destination.
5. The system must have single-board computer (SBC).	Data processing and running programs.
6. The system shall support IEEE 802.11 wireless LAN.	Using the wireless network to control and visualize the system remotely.
7. The system shall have a 360-degree laser sensor.	Build 2D SLAM mapping.
8. The R/C car shall have an operating system.	Connecting hardware and software to collect the data and give it commands.
9. The system shall estimate the shortest path to the target point automatically.	Estimate the shortest path to the target point by using efficient algorithms.

### Section 2.2 Full Requirement Specification

Engineering Requirements	Corresponding Marketing Requirement
1. The system shall have RGB-D camera.	2,1
2. The car must use steering system.	2,3
3. The car chassis shall between 1/18 to 1/8 scales.	4
4. The components shall operate on batteries for 30 minutes.	1,4

5. The system must have single-board computer (SBC).	2,3,5
6. The system shall support IEEE 802.11 wireless LAN.	5
7. The system shall have a 360-degree laser sensor.	1,2
8. The car shall have operating system.	2,3,5
9. The system shall estimate the shortest path to the target point automatically.	1,2,3,4
<b>Marketing requirements:</b> <ol style="list-style-type: none"> <li>1. The components shall be powered by battery.</li> <li>2. The system shall be able to detect obstacles and avoid them.</li> <li>3. The system shall move from an initial point to a target point.</li> <li>4. The system shall work at indoor environment.</li> <li>5. The system shall be controlled remotely.</li> </ol>	

## Chapter 3      The Design

Our project design concept is based on the use of SLAM on ROS mainly. To achieve these concepts, we need an operating system and a mobile computer (Processing Unit) on R/C car chassis to give a mobility feature and independent control. Also, collecting data to build 2D and 3D mapping of the environment and localize the R/C car, we used RGB-D camera and LIDAR scanner appropriate choice for this situation. After collecting and processing the data, the control application moves the car through an Embedded or Movement system.

### Section 3.1 Overview of the system

The prototype design that we selected from research survey, subdivided into these components:

- Data Collection.
- Processing Unit.
- Embedded or Movement system.
- ROS environment.

#### 3.1.1 Data Collection

First of all, we select RPLIDAR A1 and Xbox360 Kinect RGB-D camera as collection of data from the environment.

##### 3.1.1.1 RPLIDAR A1:

is based on laser triangulation ranging principle and runs clockwise to perform a 360-degree omnidirectional laser range scanning for its surrounding environment and then generate an outline map for the environment. Reach 5.5Hz scan rate, and provides 0.125 millisecond sample duration, 360-degree angular range, and 0.15 – 6m range. Used to build 2D mapping with 360° angular range at indoor environment [18].



Figure 10: RPLIDAR A1 [18].

#### Experiment:

After you setup the project on the R/C car that we refer in Chapter 4. Connect RPLIDAR with Raspberry pi and run this command:

```
roslaunch rplidar_ros rplidar.launch
```

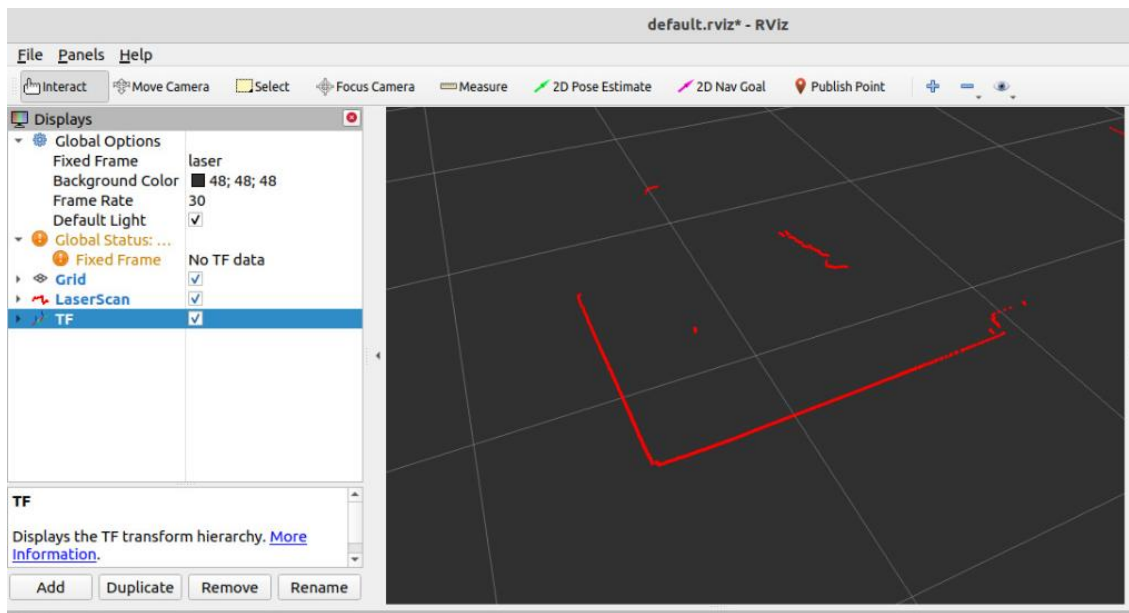


Figure 11: The result show 2D map with LaserScan on RVIZ of the room.

This picture shows laser scan points gathered from RPLIDAR on Rviz visualization to visualize the state of the R/C car.

#### 3.1.1.2 Xbox360 Kinect RGB-D camera:

Appropriate for us, have 50 cm to 5 m range,  $57^{\circ}\text{H} \times 43^{\circ}\text{V}$  FOV, and  $320 \times 240$  16-bit depth at 30 frames/sec RGB and Depth image resolution, and infrared (IR) emitter and an IR depth sensor give the Kinect its depth measuring capabilities. Which is appropriate for indoor environment, and compatible with ROS. The camera hardware will need 12v power with USB port adapter to connect it with a computer.



Figure 12: Xbox360 Kinect RGB-D camera [19]..

#### Experiment:

Connect USB cable of Kinect with Raspberry pi and run this command:

```
roslaunch freenect _launch freenect.launch depth_registration:=true
```

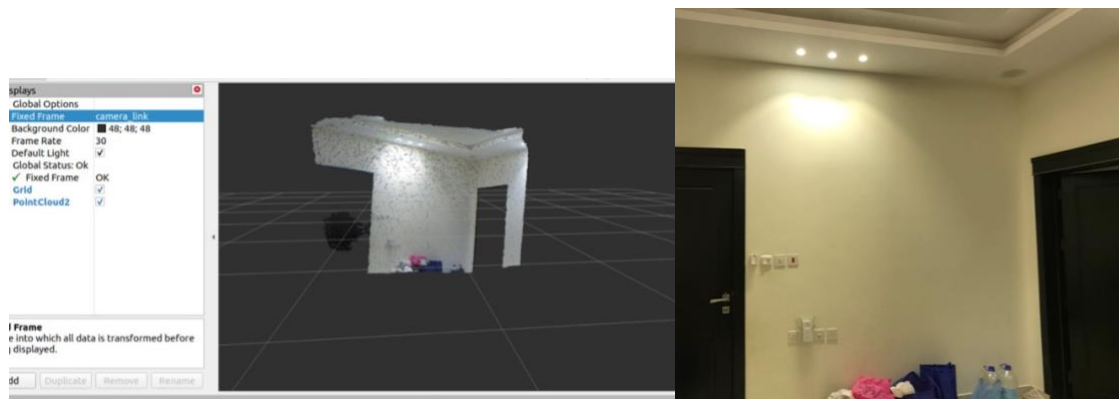


Figure 13: The result show 2D map with PointCloud2 on RVIZ in left picture and picture of the room in the right.

This picture shows points cloud and depth image of the room and visualize it on Rviz, used by different applications in R/C car.

### 3.1.2 Processing Unit

Processing Unit is based on Raspberry pi 4B small single board computer that run a variety of Linux-based operating systems, including Raspberry Pi OS, Ubuntu Mate, Snappy Ubuntu Core, Kodi-based media centers OSMC and LibreElec, and many more. Therefore, it has 64-bit ARM-Cortex A72 CPU running at 1.5GHz, and 2 Gigabyte RAM, and 802.11 b/g/n/ac Wireless LAN, 2 port USB2 and 2 port USB3, and support GPIO pins [20].

Although there are better ones such as Nvidia Jetson nano, but due to its unavailability. Used Raspberry Pi 4B rather than Nvidia Jetson nano to install ROS noetic distribution on Ubuntu 20.04 operating system.



Figure 14: Raspberry Pi 4B [20].

### 3.1.3 Movement system

Without electricity thought it's not going to do much good and easiest way to make it move for DC motor at least to apply the appropriate is supply voltage. Part of the

reason we can't just plugin motor straight into Raspberry Pi or an Arduino is that motors require a relatively high voltage and high current compared to what the pins on these boards can handle. Instead, we use a motor driver to takes a low voltage and low current signal from controller and uses power supply to simplify it creating a higher voltage and higher current feed to drive the motors. This voltage signal called PWM drives forward, and back.

Here are the needs of motor controller to generate PWM and set the enough signals. The DC motor Encoder shown in Figure 15, gives feedback to the motor controller with speed information and positioning.

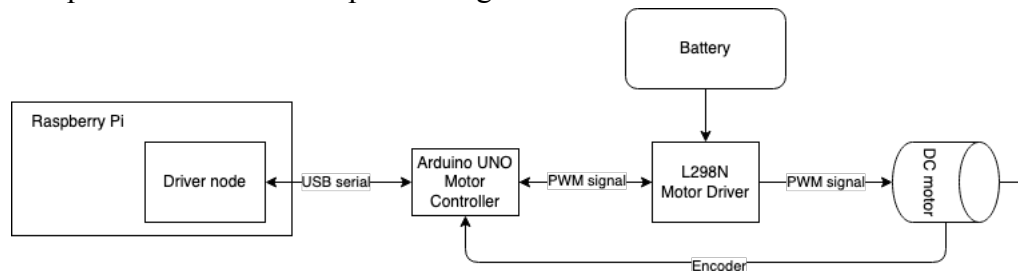


Figure 15: Movement system.

The R/C car chassis have a DC motor (without encoder) and Servo motor connected to L298N Motor driver as in Figure 18 to control the speed and spinning direction of DC motor and Servo motor. In addition, there is PWM technique we referred in previously to control speed, and H-Bridge to control the spinning direction.

- **PWM:** DC motor can be controlled by changing its average value of the input voltage which is referred to as the Duty Cycle. The higher Duty Cycle, the higher applied voltage to DC motor and so forth. The Bellow illustrate Duty Cycle.

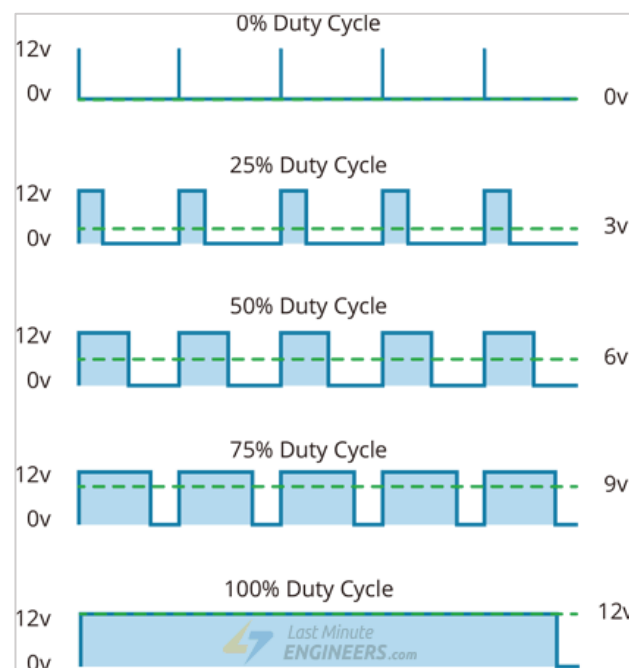


Figure 16: PWM technique with various Duty Cycles and average voltages [21].



- **H-Bridge:** The spinning direction of a DC motor can be controlled by changing the polarity of its input voltage. Closing two specific switches at the same time in reverses the polarity of the voltage applied to the motor. This causes a change in the spinning direction of the motor.

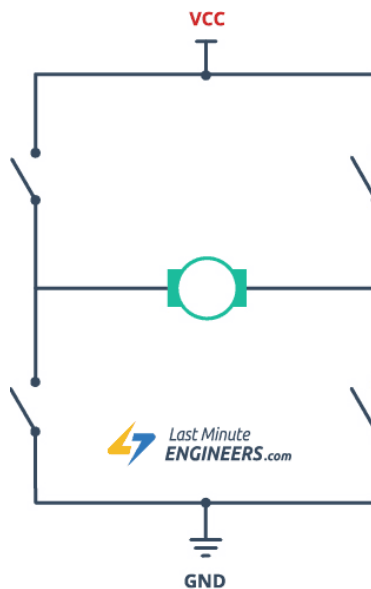


Figure 17: H-Bridge circuit [21].

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection to a computer or power supply input by 7-12V, and reset button. Furthermore, Arduino UNO connected to Raspberry Pi by using USB cable, and rosserial protocol to allow the messages flow in and out of Arduino [22] .

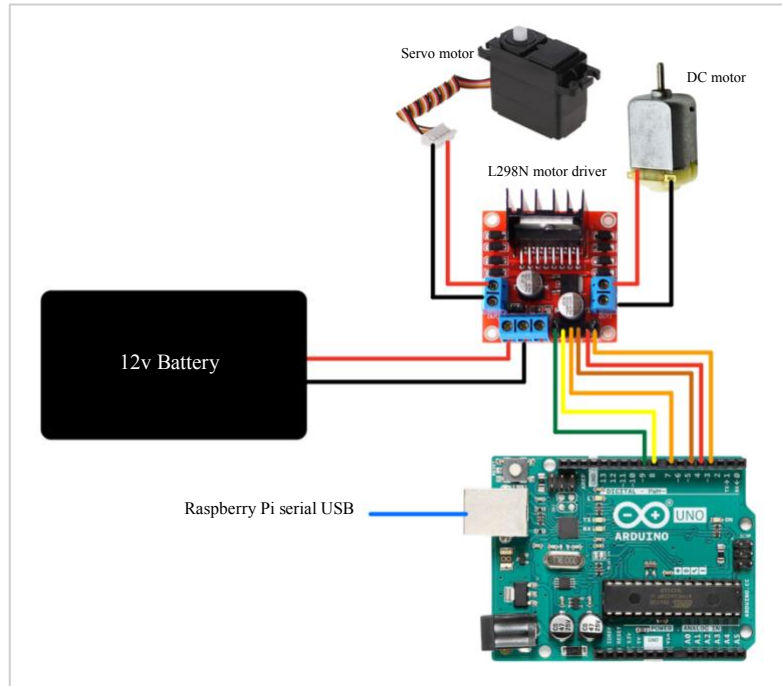


Figure 18: Movement system wiring.

### 3.1.4 ROS environment

Navigation is the movement of the robot to a defined destination. It is important to find the optimized route among the various routing options, and to avoid obstacles such as walls. To navigate using a laser for localization and obstacle avoidance with Kinect camera, you need to do a map of the environment you want to move through.

#### 3.1.4.1 Mapping

Mapping gives a robot that has a perfect estimate of the position, and a sequence of measurements to determine the map of the environment. The map enables the robot to localize itself and receive commands to navigation to a specific location within the map. To do that we need some information like odometry, laser scanner, and point cloud. Also, there are many kinds of maps could we use with ROS such as Gmapping, Hector Mapping, and RTAB-Map:

- **Gmapping:** is package provides laser-based SLAM (Simultaneous Localization and Mapping). You can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data (odometry) collected by a mobile robot [23].
- **Hector Mapping:** is a SLAM approach that can be used without odometry. It leverages the high update rate of modern LIDAR and provides 2D pose estimates. The system has successfully been used on Unmanned Ground Robots, Unmanned Surface Vehicles, Handheld Mapping Devices, and logged data from quadrotor UAVs [24].
- **RTAB-Map:** is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. The loop closure

detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. Furthermore, can be used alone with a handheld Kinect, a stereo camera, or a 3D lidar for 6DoF mapping, or on a robot equipped with a laser rangefinder for 3DoF mapping [25].

We observed the appropriate map is RTAB-Map, it uses RGB-D camera and laser scanner.

#### 3.1.4.2 URDF

There may be many different software components that need to know about the physical characteristics of the robot. For consistency and simplicity, it is good to keep all this information in one common location, where any code can reference it. In ROS, this called robot description and the information is stored in a URDF (Unified Robot Description Format) file [26].

Additionally, URDF file describes a robot as a tree of links, that are connected by joints. The links represent the physical components of the robot, and the joints represent how one link moves relative to another link, effectively defining the location of the links in space.

This is passed to robot\_state\_publisher node which makes the data available on the /robot\_description topic, and broadcasts the appropriate transforms. If there are any joints that move, robot\_state\_publisher will expect to see the input values published on the /joint\_states topic, and while doing our initial tests we can use the joint\_state\_publisher\_gui to fake those values.

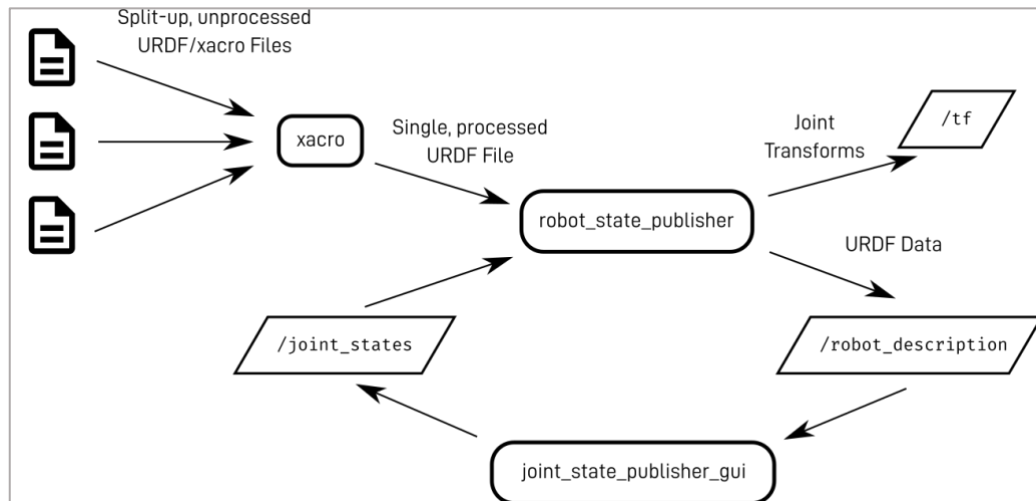
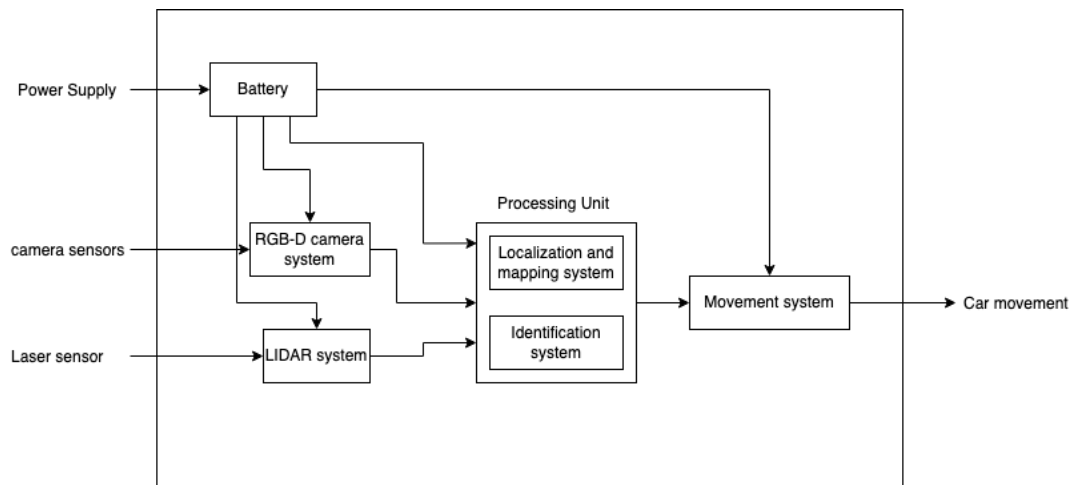


Figure 19: Nodes published to visualize robot description in Rviz simulation.

## Section 3.2 Functional Specifications

### 3.2.1 Level 1 design:



Battery	
Inputs	Power supply.
Outputs	-Processing Unit. -RGB-D Camera. -LIDAR system. -Movement system.
Functionality	Provide enough power to all modules.

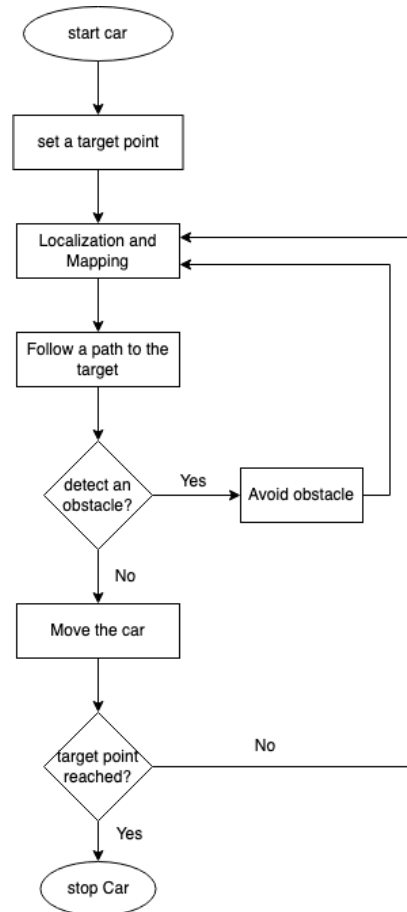
Localization and Mapping system	
Inputs	-Battery. -LIDAR system.
Outputs	- Movement system.
Functionality	Localize and build a map by collected the data from LIDAR.

Identification system	
Inputs	-RGB-D Camera system.
Outputs	-Movement system.
Functionality	Identify the obstacles.

Processing Unit	
Inputs	-RGB-D camera system. -LIDAR system. -Battery.
Outputs	-Movement system.
Functionality	Data collection, processing and giving commands to movement system.

Movement system	
Inputs	- Processing Unit.
Outputs	-car movement.
Functionality	Get the commands from the Processing Unit and give an action to the car.

### 3.2.2 Workflow diagram:



Process	Description
Start the Car.	Turn on the Car.
Set a target Point.	Positioning and navigate a target point.
Localization and Mapping.	Find the current position and mapping the environment by LIDAR sensor to locate obstacles.
Follow the path to the target point.	Estimate the path to the target point by using algorithms efficiently.

Avoid the obstacle.	Avoid the obstacle by going left or right, forward, or backward.
Move the car.	Going forward until to reach the target point.
Stop the car.	Turn off the car after reaching the target point.

## Chapter 4 Implementation and Verification

The R/C car project demonstration partitioning into two sections:

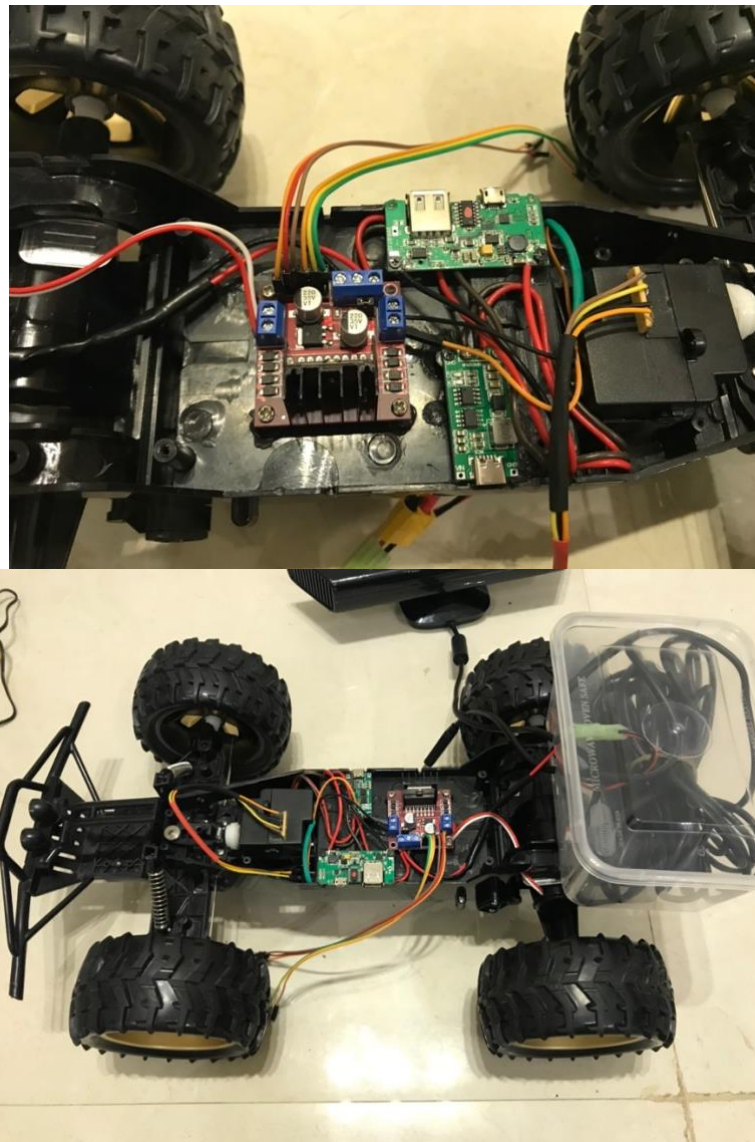
- Section 1: R/C car chassis.
- Section 2: Software environment.

### Section 1: R/C car chassis

After these components are built, it's time to put them together in R/C car chassis.

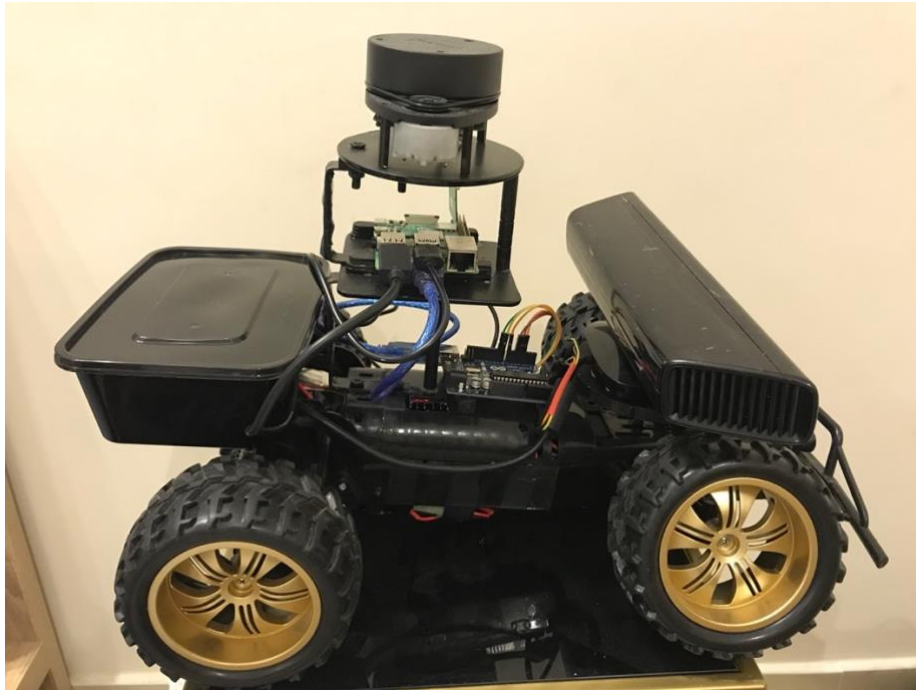
The R/C car was built in several stages:

- 1- Making a portable rechargeable power system with 12v Kinect camera power and USB-A 5v for Raspberry Pi power, USB type-c used to recharge the battery.
- 2- Movement system integration with car chassis.



*Figure 20: power and movement systems.*

- 3- Install the Data Collection components above the chassis:



*Figure 21: Final R/C car with all components from right side.*



*Figure 22: final R/C car with all components from front side.*



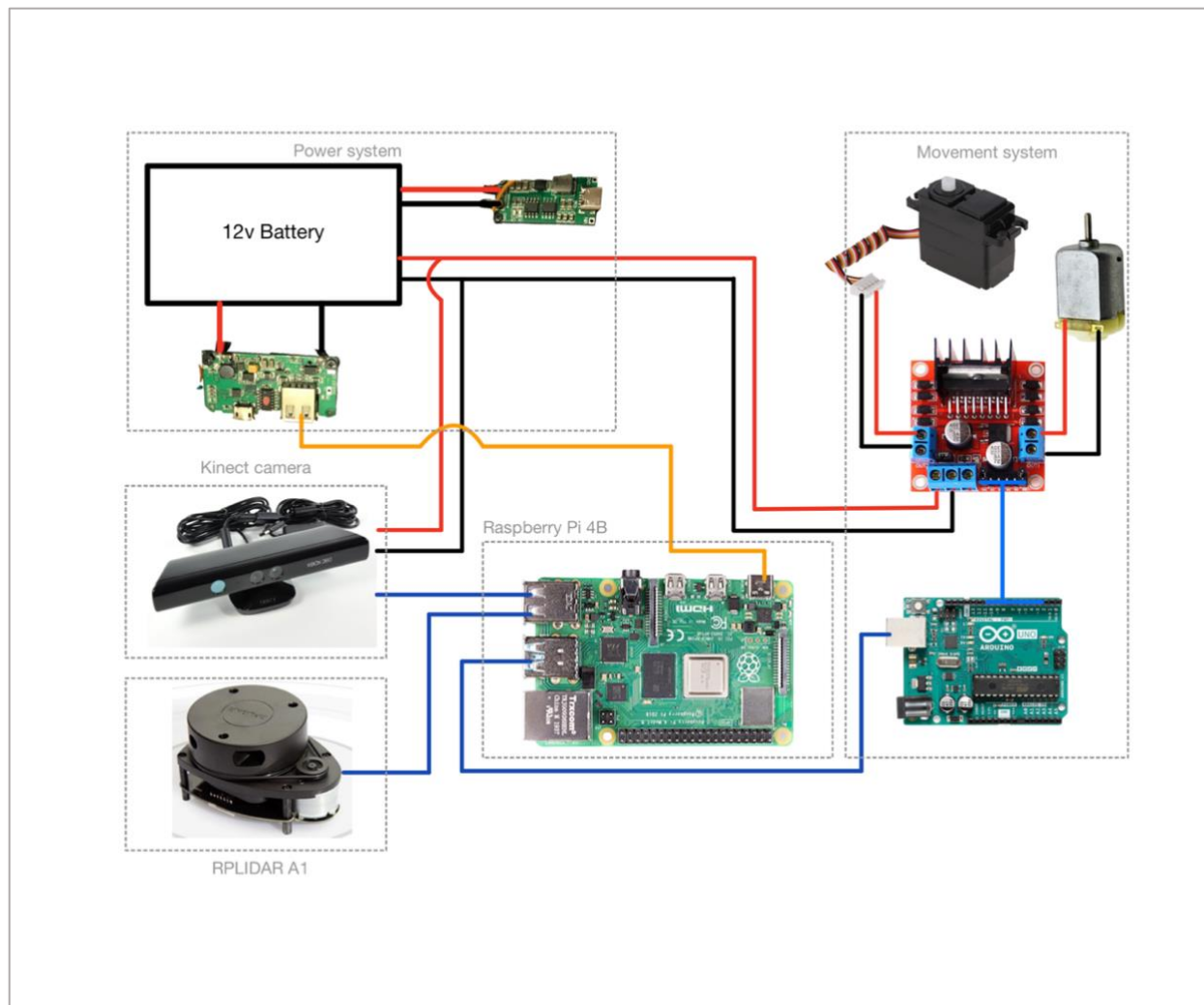


Figure 23: Wiring all R/C car components.

## Section 2: Software environment

1. Download Ubuntu 20.4 on your computer: [click here](#).
2. Download Raspberry pi imager to burn ubuntu 20.4 on 64Gb SD card: [click here](#).
3. Install Ubuntu 20.4 on Raspberry Pi: [click here](#).
4. Enable SSH server on Raspberry Pi and SSH client on your computer to control remotely: [click here](#).
5. When you get “Error Permission denied (publickey)”: [click here](#).
6. Got this error message:

```
[user@hostname ~]$ ssh root@pong
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
```

Fix this problem by this [link](#).

7. Install ROS Noetic distribution on computer and Raspberry Pi: [click here](#).

8. Setup xbox360 Kinect: [click here](#).
9. Setup RPLIDAR A1 sensor: [click here](#).
10. Learning how tf package could work: [click here](#).
11. Learning URDF to build your robot model on the Rviz visualization: [click here](#).
12. Build URDF file of Data collection components with [Kinect\\_description](#), [rplidar\\_model](#).
13. Our R/C car URDF file combined of [turtlebot3](#), [rosbot](#), [articubot\\_one](#), and [RoboCar](#).
14. Connect Kinect camera and RPLIDAR scanner on Raspberry Pi by USB cable.
15. Open new terminal on your computer and run "ssh <username>@<Raspberry Pi IP address>" use SSH client on your desktop to control Raspberry Pi.
16. To enable data transition between Raspberry Pi and the computer you need to choose what is the master by using these instructions: [click here](#).
17. Open new terminal on Raspberry Pi terminal and run this command: `roslaunch car_description sensors.launch model:='$(find car_description)/urdf/myRobot.xacro'` used to launch RPLIDAR and Kinect sensor.  
Then run `roslaunch car_description DisplayRobot.launch model:='$(find car_description)/urdf/myRobot.xacro'` command in new terminal on the computer.  
Will show you RobotModel, LaserScan, and PointCloud2 combined in Rviz visualization.

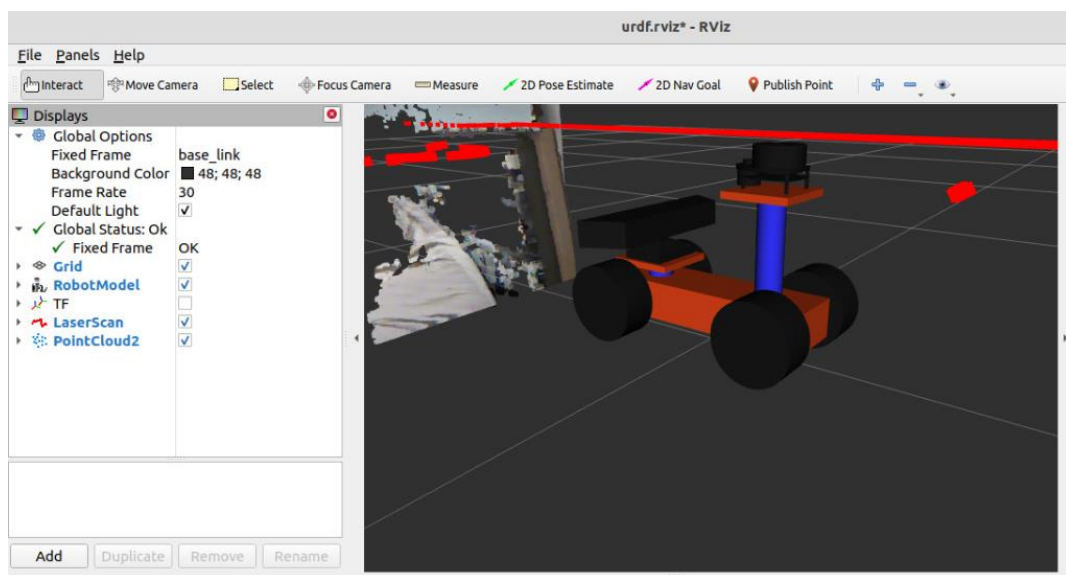


Figure 24: RobotModel of URDF visualization.

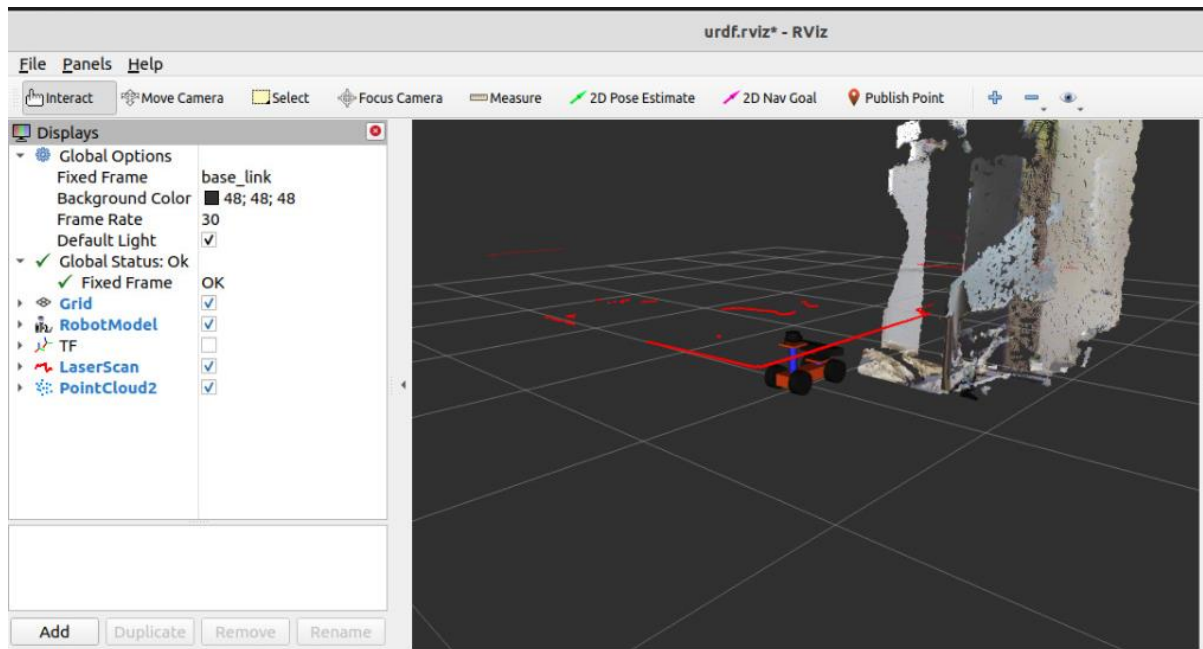


Figure 25: LaserScan of RPLIDAR visualization.

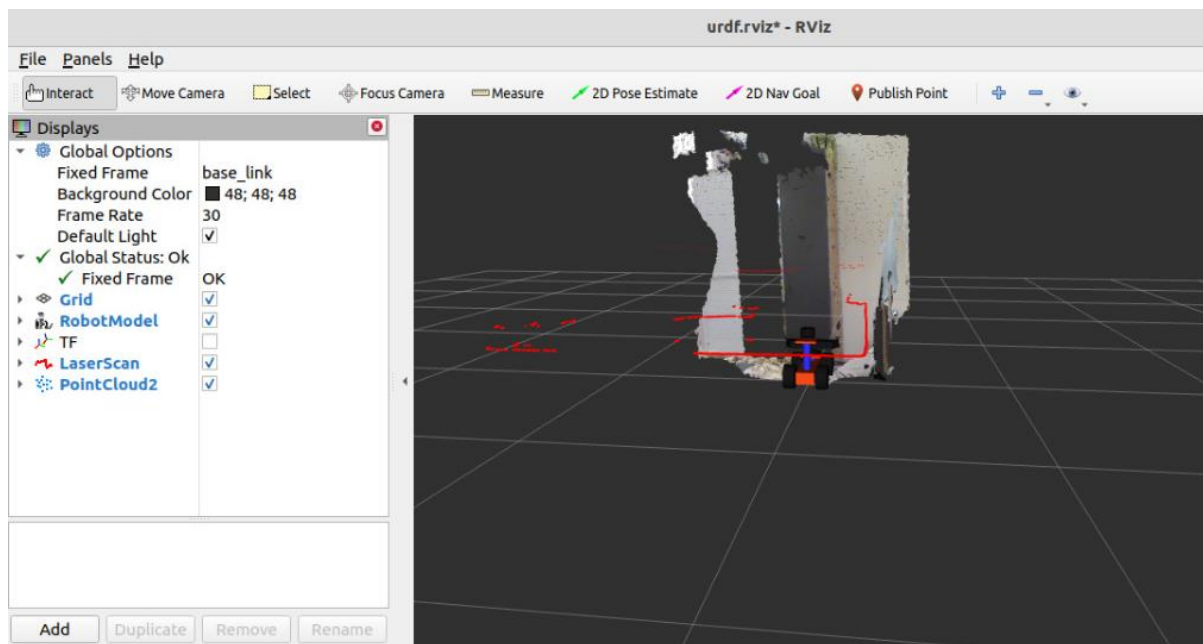


Figure 26: PointCloud2 of Kinect visualization.

This project is not complete, it will collect the data from RPLIDAR and Kinect camera to display them on Rviz to use in many applications.

To achieve the requirements of the project, you need to use one of the mapping libraries had been mentioned before to build the map of the environment, as well as you need to develop an application that identifies and avoids objects.

As for Embedded system, you need to use rosserial library on Arduino and Raspberry Pi to send instructions like go forward, backward, left, and right. as well as there are four servo motor wires should you find out how to use it.

## **Conclusion**

As you have seen in this report we provided the motivation, objective statement, and research survey that helped us to take what are the technologies used and verificate the requirements of this project.

We faced many challenges, including setup the environment and choosing the appropriate packages to build the project. There are many libraries due to the presence of a large community of programmers, which makes it difficult to deal with them and link them to each other.

To improve this project, I recommend replacing Raspberry Pi with a more powerful single-board computer such as Nvidia Jetson nano. As well as replacing the Kinect camera with a higher resolution RGB-D camera with a larger Field of View, and its large size, and having a special power port.

## References

- [1] "The top 10 causes of death," World Health Organization, 2020. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>.
- [2] "Najm announces launching the Comprehensive Auto Insurance Management Solutions center "CAMS"," Najm, 30 January 2019. [Online]. Available: <https://najm.sa/sites/en/News/55>.
- [3] "Automated Vehicles for Safety," [Online]. Available: <https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety#:~:text=The%20continuing%20evolution%20of%20automotive,can't%20do%20it%20ourselves..>
- [4] "SLAM (Simultaneous Localization and Mapping)," mathworks, [Online]. Available: <https://www.mathworks.com/discovery/slam.html>.
- [5] Z. Shishun, Z. Longyu and T. Wenbing, "Survey and Evaluation of RGB-D SLAM," 21 January 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9330596>.
- [6] M. Lili, Y. Pantao, Z. Yuchen, C. Kai, W. Fangfang and Q. Nana, "Research on SLAM Algorithm of Mobile Robot Based on the Fusion of 2D LiDAR and Depth Camera," 26 August 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/9178302#full-text-header>.
- [7] P. Yoonseok, C. Hanchoul and J. Leon, ROS Robot Programming (in English), ROBOTIS Co.,Ltd., 2017.
- [8] "Robot Operating System (ROS)," [Online]. Available: <https://www.ros.org/>.
- [9] "RGB-D Sensors and 3D Reconstruction," [Online]. Available: [https://iee-sensors.org/wp-content/uploads/2019/05/SJ\\_Special\\_Issue\\_RGB-D\\_Sensors.pdf](https://iee-sensors.org/wp-content/uploads/2019/05/SJ_Special_Issue_RGB-D_Sensors.pdf).
- [10] G. Sara, L. Manuel, R. Jennifer and G. Anton, "Concept of an Automotive LiDAR Target Simulator for Direct Time-of-Flight LiDAR," 8 AUGUST 2015. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9618816>.
- [11] G. Ziming, C. Baigen, J. Wei and W. Jian, "Feature-based detection and classification of moving objects using LiDAR sensor," 26 March 2019. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2018.5291>.
- [12] Z. Jiaheng, H. Shoudong, Z. Liang, C. Yongbo and L. Xiao, "Conic Feature Based Simultaneous Localization and Mapping in Open Environment via 2D Lidar," 28 November 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/8917627>.
- [13] "Turtlebot3," [Online]. Available: <https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/#run-slam-node>.
- [14] S. Yogamani, C. Hughes, J. Horgan, G. Sistu and S. Chennupati, "WoodScape: A multi-task, multi-camera fisheye dataset for autonomous driving," 27 October 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/9008254>.
- [15] C. David, E. Jakob and C. Daniel , "Large-scale direct SLAM for omnidirectional cameras," 2015. [Online]. Available: <https://ieeexplore.ieee.org/document/7353366/authors#authors>.
- [16] "Instrumentation of an array of ultrasonic sensors and data processing for unmanned aerial vehicle (UAV) for teaching the application of the kalman filter," 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050915037217>.

- [17] "How Does A Servo Motor Work?," [Online]. Available: <https://www.electricaleasy.com/2015/01/how-does-servo-motor-work.html>.
- [18] "RPLIDAR A1," [Online]. Available: <https://www.slamtec.com/en/Lidar/A1>.
- [19] "Xbox 360 Kinect Teardown - iFixit," iFixit, 2010. [Online]. Available: <https://www.ifixit.com/Teardown/Xbox+360+Kinect+Teardown/4066>.
- [20] "Raspberry pi 4B Specification," Raspberry Pi Foundation, [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
- [21] "Interface L298N DC Motor Driver Module with Arduino," [Online]. Available: <https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/>.
- [22] "rosterial - ROS Wiki," [Online]. Available: <http://wiki.ros.org/rosterial>.
- [23] "gmapping - ROS Wiki," [Online]. Available: <http://wiki.ros.org/gmapping>.
- [24] "hector\_mapping - ROS Wiki," [Online]. Available: [http://wiki.ros.org/hector\\_mapping](http://wiki.ros.org/hector_mapping).
- [25] "RTAB-Map | Real-Time Appearance-Based Mapping," [Online]. Available: <https://introlab.github.io/rtabmap/>.
- [26] "urdf - ROS Wiki," [Online]. Available: <http://wiki.ros.org/urdf>.
- [27] "ASTRA," [Online]. Available: <https://shop.orbbec3d.com/Astra>.
- [28] J. Dong-Won, L. Zhong-Soo, K. Byung-Geuk and K. Nak-Ku , "Multi-channel ultrasonic sensor system for obstacle detection of the mobile robot," 2007. [Online]. Available: <https://ieeexplore.ieee.org/document/4406722/authors#authors>.

## Appendix A      The Project Plan

### Section 4.1 Work Breakdown Structure

<u>ID</u>	<u>Activity</u>	<u>Description</u>	<u>Check points</u>	<u>Duration (days)</u>	<u>People</u>	<u>Predecessors</u>
<b>1</b>	<b>Problem statement</b>	<b>Writing the problem statement.</b>		<b>35</b>	<b>Faisal Almalki</b>	
1.1	Need statement	Writing need statement.		7	Faisal Almalki	
1.2	Objective	Writing objective statement.		7	Faisal Almalki	1.1
1.3	Research survey	Researching and writing relevant technologies.		14	Faisal Almalki	1.2
1.4	User needs	Collect and writing user needs.		7	Faisal Almalki	1.4
<b>2</b>	<b>Requirement specification</b>	<b>Determine the requirements dependent on user needs.</b>		<b>15</b>	<b>Faisal Almalki</b>	<b>1</b>
2.1	Engineering Requirements	Determine standard technologies or engineering requirements dependent on user needs.		10	Faisal Almalki	1.4
2.2	Full Requirement Specification	Mapping engineering requirements to user needs.		5	Faisal Almalki	1.4
<b>3</b>	<b>The Design</b>			<b>77</b>	<b>Faisal Almalki</b>	<b>2</b>
3.1	Overview of the system	describes the system's concept and how it will satisfy the requirements.		<b>20</b>	<b>Faisal Almalki</b>	<b>2.2</b>
3.2	Level 1	diagram illustrating the system architecture.		23	Faisal Almalki	3.1
3.3	Workflow diagram.	describes how the system works.		34	Faisal Almalki	3.2
4	Cost Estimation.	approximate cost estimate for the design project.		3	Faisal Almalki	3.1



## Section 4.2 Gantt Chart

0	Project Work Plan		202 days	0%	Mon 1/17/22 8:00	Tue 10/25/22 5:0	1,656 hr	
1	Researching	Faisal Almalki	14 days	0%	Mon 1/17/22 8:00 AM	Thu 2/3/22 5:00 PM	112 hrs	
2	The Problem statement		35 days	0%	Fri 2/4/22 8:00 AM	Thu 3/24/22 5:00 P	280 hrs	
3	Need statement	Faisal Almalki	7 days	0%	Fri 2/4/22 8:00 AM	Mon 2/14/22 5:00 PM	56 hrs	1
4	Objective statement	Faisal Almalki	7 days	0%	Tue 2/15/22 8:00 AM	Wed 2/23/22 5:00 PM	56 hrs	3
5	Research survey	Faisal Almalki	14 days	0%	Thu 2/24/22 8:00 AM	Tue 3/15/22 5:00 PM	112 hrs	4
6	User needs	Faisal Almalki	7 days	0%	Wed 3/16/22 8:00 AM	Thu 3/24/22 5:00 PM	56 hrs	5
7	Requirements specification		15 days	0%	Fri 3/25/22 8:00 A	Thu 4/14/22 5:00 P	120 hrs	2
8	Engineering Requirements	Faisal Almalki	10 days	0%	Fri 3/25/22 8:00 AM	Thu 4/7/22 5:00 PM	80 hrs	6
9	Full Requirement Specification	Faisal Almalki	5 days	0%	Fri 4/8/22 8:00 AM	Thu 4/14/22 5:00 PM	40 hrs	8
10	The Design		77 days	0%	Fri 4/15/22 8:00 A	Mon 8/1/22 5:00 P	616 hrs	7
11	Overview of the system	Faisal Almalki	14 days	0%	Fri 4/15/22 8:00 AM	Wed 5/4/22 5:00 PM	112 hrs	
12	Level 1 design	Faisal Almalki	3 days	0%	Thu 5/5/22 8:00 AM	Mon 5/9/22 5:00 PM	24 hrs	11
13	Functional Specifications	Faisal Almalki	30 days	0%	Tue 5/10/22 8:00 AM	Mon 6/20/22 5:00 PM	240 hrs	12
14	buy the Hardware	Faisal Almalki	30 days	0%	Tue 6/21/22 8:00 AM	Mon 8/1/22 5:00 PM	240 hrs	13
15	Hardware installation		27 days	0%	Tue 8/2/22 8:00 A	Wed 9/7/22 5:00 P	256 hrs	14
16	install the car body	Faisal Almalki	5 days	0%	Tue 8/2/22 8:00 AM	Mon 8/8/22 5:00 PM	40 hrs	
17	LIDAR and stereo camera insatllation	Faisal Almalki	5 days	0%	Tue 8/9/22 8:00 AM	Mon 8/15/22 5:00 PM	40 hrs	16
18	install the Hardware drivers	Faisal Almalki	5 days	0%	Tue 8/16/22 8:00 AM	Mon 8/22/22 5:00 PM	40 hrs	17
19	ROS Software		12 days	0%	Tue 8/23/22 8:00 A	Wed 9/7/22 5:00 P	96 hrs	18
20	Prepare the software environment	Faisal Almalki	5 days	0%	Tue 8/2/22 8:00 AM	Mon 8/8/22 5:00 PM	40 hrs	
21	Select the SLAM algorithms.	Faisal Almalki	5 days	0%	Tue 8/23/22 8:00 AM	Mon 8/29/22 5:00 PM	40 hrs	20
22	Neural network learning	Faisal Almalki	7 days	0%	Tue 8/30/22 8:00 AM	Wed 9/7/22 5:00 PM	56 hrs	21
23	Testing		19 days	0%	Thu 9/8/22 8:00 A	Tue 10/4/22 5:00 P	152 hrs	
24	Develop unit test plans using product	Faisal Almalki	4 days	0%	Thu 9/8/22 8:00 AM	Tue 9/13/22 5:00 PM	32 hrs	19
25	Develop integration test plans using product specifications	Faisal Almalki	15 days	0%	Wed 9/14/22 8:00 AM	Tue 10/4/22 5:00 PM	120 hrs	24
26	Documentation		15 days	0%	Wed 10/5/22 8:00	Tue 10/25/22 5:00	120 hrs	23
27	Implementation and Verification	Faisal Almalki	5 days	0%	Wed 10/5/22 8:00 AM	Tue 10/11/22 5:00 PM	40 hrs	
28	The project Plan	Faisal Almalki	3 days	0%	Wed 10/12/22 8:00 A	Fri 10/14/22 5:00 PM	24 hrs	27
29	Data sheets	Faisal Almalki	7 days	0%	Mon 10/17/22 8:00 A	Tue 10/25/22 5:00 PM	56 hrs	28

Figure 7: Gantt chart



Figure 8: Timeline



### Section 4.3 Cost Estimation

Component	price (\$)	Quantity	Total Price (\$)
LIDAR 360 Degree	155	1	155
Xbox360 Kinect	30	1	30
1:10 R/C Car	60	1	60
Raspberry Pi 4B 4G	75	1	75
64G SD card	15	1	15
Arduino UNO	18	1	18
Motor Driver	15	1	15
12v Battery	150	1	150
other	50		50
Total cost (\$)			568

## Appendix B Program Code

Project files are in [faisal-20](#) GitHub profile with the RC-car repository. Some of the files have been installed inside the catkin\_ws folder from other sources we mentioned before. Here, we will put some important files you should look at it.

*car\_description/launch/display.launch:*

```
<launch>

<arg name="model" default="$(find car_description)/urdf/myRobot.xacro"/>
<arg name="gui" default="true" />
<arg name="rvizconfig" default="$(find car_description)/rviz/urdf.rviz" />

<param name="robot_description" command="$(find xacro)/xacro $(arg model)" />

<node if="$(arg gui)" name="joint_state_publisher" pkg="joint_state_publisher_gui"
type="joint_state_publisher_gui" />
<node unless="$(arg gui)" name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher"
/>

<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />

<!-- Lidar Data Publisher Using RPLIDAR from Slamtec -->
<node name="rplidarNode"      pkg="rplidar_ros" type="rplidarNode" output="screen">
<param name="serial_port"    type="string" value="/dev/ttyUSB0"/>
<param name="serial_baudrate" type="int" value="115200"/><!--A1/A2 -->
<!--param name="serial_baudrate" type="int" value="256000"--><!--A3 -->
<param name="frame_id"       type="string" value="laser"/>
<param name="inverted"       type="bool" value="false"/>
<param name="angle_compensate" type="bool" value="true"/>
</node>

<!-- kinect launch -->
<include file="$(find freenect_launch)/launch/freenect.launch"/>

<node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />

</launch>
```

*car\_description/urdf/myRobot.xacro:*

```
<?xml version="1.0"?>
<robot name="myRobot" xmlns:xacro="http://ros.org/wiki/xacro">

  <!-- Import xacro files -->
  <xacro:include filename="$(find car_description)/urdf/materials.xacro"/>
  <xacro:include filename="$(find car_description)/urdf/rplidar.xacro"/>
  <xacro:include filename="$(find car_description)/urdf/kinect.xacro"/>
  <xacro:include filename="$(find car_description)/urdf/inertial_macros.xacro"/>

  <!-- variables or properties -->
  <xacro:property name="chassis_length" value="0.35"/>
  <xacro:property name="chassis_width" value="0.09"/>
  <xacro:property name="chassis_height" value="0.04"/>
  <xacro:property name="chassis_mass" value="1"/>

  <xacro:property name="wheel_radius" value="0.06"/>
  <xacro:property name="wheel_thickness" value="0.055"/>
  <xacro:property name="wheel_mass" value="0.05"/>
  <xacro:property name="wheel_offset_x" value="0.226"/>
  <xacro:property name="wheel_offset_y" value="0.1485"/>
  <xacro:property name="wheel_offset_z" value="0.01"/>

  <xacro:property name="rear_wheel_base_length" value="0.16"/>
  <xacro:property name="rear_wheel_base_radius" value="0.02"/>

  <xacro:property name="front_wheel_base_length" value="0.16"/>
  <xacro:property name="front_wheel_base_radius" value="0.02"/>

  <!-- materials -->
  <material name="white">
    <color rgba="1 1 1 1" />
  </material>

  <material name="orange">
    <color rgba="1 0.3 0.1 1"/>
  </material>
```

```

<material name="blue">
  <color rgba="0.2 0.2 1 1"/>
</material>

<material name="black">
  <color rgba="0.1 0.1 0.1 1"/>
</material>

<material name="red">
  <color rgba="1 0 0 1"/>
</material>

<!-- ===== base link ===== -->
<link name="base_link">
  <visual>
    <origin xyz="0 0 ${wheel_radius}"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.01 0.01 0.01 "/>
    </geometry>
    <material name="white"/>
  </visual>
  <collision>
    <origin xyz="0 0 ${wheel_radius}"/>
    <geometry>
      <box size="0.01 0.01 0.01"/>
    </geometry>
  </collision>
</link>

<!-- ===== Chassis =====>
<joint name="chassis_joint" type="fixed">
  <parent link="base_link"/>
  <child link="chassis"/>
  <origin xyz="0 0 ${wheel_radius}"/>
</joint>

<link name="chassis">

```

```

<visual>
  <origin xyz="0 0 0"/>
  <geometry>
    <box size="{chassis_length} {chassis_width} {wheel_radius}"/>
  </geometry>
  <material name="orange"/>
</visual>

<collision>
  <origin xyz="{chassis_length} 0 {wheel_radius}"/>
  <geometry>
    <box size="{chassis_length} {chassis_width} {chassis_height}"/>
  </geometry>
</collision>

<xacro:inertial_box mass="0.5" x="{chassis_length}" y="{chassis_width}" z="{chassis_height}">
  <origin xyz="{chassis_length} 0 {chassis_height/2}" rpy="0 0 0"/>
</xacro:inertial_box>

</link>

<gazebo reference="chassis">
  <!--Stiffness -->
  <kp>1000000.0</kp>
  <!--Dampening-->
  <kd>0.1</kd>
  <dampingFactor>0</dampingFactor>
  <material>Gazebo/White</material>
  <selfCollide>true</selfCollide>
  <turnGravityOff>false</turnGravityOff>
  <mu1 value="0.1"/>
  <mu2 value="0.1"/>
  <fdir1 value="0 0 0"/>
</gazebo>

<!-- ===== rear wheel base =====>
<joint name="rear_wheel_base_joint" type="fixed">
  <parent link="chassis"/>
  <child link="rear_wheel_base"/>
  <origin xyz="-{chassis_length/2-rear_wheel_base_radius-0.02} 0 0" rpy="-{pi/2} 0 0"/>
</joint>

```

```

<link name="rear_wheel_base">
  <visual>
    <geometry>
      <cylinder length="${rear_wheel_base_length}" radius="${rear_wheel_base_radius}"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <origin xyz="0 0 0"/>
    <geometry>
      <cylinder length="${rear_wheel_base_length}" radius="${rear_wheel_base_radius}"/>
    </geometry>
  </collision>
</link>

<gazebo reference="rear_wheel_base">
  <material>Gazebo/Black</material>
</gazebo>

<!-- =====rear left wheel===== -->
<joint type="continuous" name="rear_left_wheel_hinge">
  <origin xyz="0 0 ${rear_wheel_base_length/2}" rpy="0 0 ${pi/2}"/>
  <child link="rear_left_wheel"/>
  <parent link="rear_wheel_base"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
  <limit effort="1" velocity="10"/>
  <joint_properties damping="5.0" friction="1.0"/>
</joint>

<link name="rear_left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_thickness}"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>

```

```

    <geometry>
      <sphere radius="${wheel_radius}"/>
    </geometry>
  </collision>

  <xacro:inertial_cylinder mass="${wheel_mass}" length="${wheel_thickness}" radius="${wheel_radius}">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:inertial_cylinder>
</link>

<gazebo reference="rear_left_wheel">
  <material>Gazebo/black</material>
</gazebo>

<!-- =====rear right wheel=====-->
<joint type="continuous" name="rear_right_wheel_hinge">
  <origin xyz="0 0 -${rear_wheel_base_length/2}" rpy="0 0 ${pi/2}"/>
  <child link="rear_right_wheel"/>
  <parent link="rear_wheel_base"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
  <limit effort="1" velocity="10"/>
  <joint_properties damping="5.0" friction="1.0"/>
</joint>

<link name="rear_right_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_thickness}"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <geometry>
      <sphere radius="${wheel_radius}"/>
    </geometry>
  </collision>
  <xacro:inertial_cylinder mass="${wheel_mass}" length="${wheel_thickness}" radius="${wheel_radius}">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:inertial_cylinder>

```

```

</link>

<gazebo reference="rear_right_wheel">
  <material>Gazebo/black</material>
</gazebo>

<!-- ===== front wheel base =====>
<joint name="front_wheel_base_joint" type="fixed">
  <parent link="chassis"/>
  <child link="front_wheel_base"/>
  <origin xyz="{chassis_length/2-front_wheel_base_radius-0.02} 0 0" rpy="-{pi/2} 0 0"/>
</joint>

<link name="front_wheel_base">
<visual>
  <geometry>
    <cylinder length="{front_wheel_base_length}" radius="{front_wheel_base_radius}"/>
  </geometry>
  <material name="black"/>
</visual>
<collision>
  <origin xyz="0 0 0"/>
  <geometry>
    <cylinder length="{front_wheel_base_length}" radius="{front_wheel_base_radius}"/>
  </geometry>
</collision>
</link>

<gazebo reference="front_wheel_base">
  <material>Gazebo/Black</material>
</gazebo>

<!-- =====front left wheel===== -->
<joint type="continuous" name="front_left_wheel_hinge">
  <origin xyz="0 0 {front_wheel_base_length/2}" rpy="0 0 {pi/2}"/>
  <child link="front_left_wheel"/>
  <parent link="front_wheel_base"/>

```



```

    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="1" velocity="10"/>
    <joint_properties damping="5.0" friction="1.0"/>
</joint>

<link name="front_left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_thickness}"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <geometry>
      <sphere radius="${wheel_radius}"/>
    </geometry>
  </collision>
  <xacro:inertial_cylinder mass="${wheel_mass}" length="${wheel_thickness}" radius="${wheel_radius}">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:inertial_cylinder>
</link>

<gazebo reference="front_left_wheel">
  <material>Gazebo/black</material>
</gazebo>

<!-- =====front right wheel=====-->
<joint type="continuous" name="front_right_wheel_hinge">
  <origin xyz="0 0 -${front_wheel_base_length/2}" rpy="0 0 ${pi/2}"/>
  <child link="front_right_wheel"/>
  <parent link="front_wheel_base"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
  <limit effort="1" velocity="10"/>
  <joint_properties damping="5.0" friction="1.0"/>
</joint>

<link name="front_right_wheel">
  <visual>

```

```

    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_thickness}" />
    </geometry>
    <material name="black" />
  </visual>
  <collision>
    <geometry>
      <sphere radius="${wheel_radius}" />
    </geometry>
  </collision>
  <xacro:inertial_cylinder mass="${wheel_mass}" length="${wheel_thickness}" radius="${wheel_radius}">
    <origin xyz="0 0 0" rpy="0 0 0" />
  </xacro:inertial_cylinder>
</link>

<gazebo reference="front_right_wheel">
  <material>Gazebo/black</material>
</gazebo>

<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/myRobot</robotNamespace>
    <legacyModeNS>true</legacyModeNS>
  </plugin>
</gazebo>

</robot>

```

*car\_description/urdf/kinect.xacro:*

```

<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">

  <xacro:include filename="$(find car_description)/urdf/materials.xacro" />

  <xacro:property name="base_length" value="0.1" />
  <xacro:property name="base_width" value="0.1" />
  <xacro:property name="base_height" value="0.01" />
  <xacro:property name="base_mass" value="0.005" />

```

```

<xacro:property name="piller2_length" value="0.03"/>
<xacro:property name="piller2_radius" value="0.02"/>

<!-- ===== Kinect ===== -->
<link name="camera_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 ${pi/2}"/>
    <geometry>
      <mesh filename="package://car_description/meshes/sensors/kinect.stl" scale="0.01 0.01 0.01"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.28 0.07 0.08"/>
    </geometry>
  </collision>
  <!-- <xacro:default_inertial mass="0.2"/> -->
</link>

<joint name="kinect_base_joint" type="fixed">
  <origin xyz="0 0 ${base_height*4}" rpy="0 0 0"/>
  <parent link="kinect_base"/>
  <child link="camera_link"/>
</joint>

<link name="kinect_base">
  <visual>
    <origin xyz="0 0 0"/>
    <geometry>
      <box size="${base_length} ${base_width} ${base_height}"/>
    </geometry>
    <material name="orange"/>
  </visual>
  <collision>
    <origin xyz="0 0 0"/>
    <geometry>

```

```

        <box size="{base_length} {base_width} {base_height}"/>
    </geometry>
</collision>
<xacro:inertial_box mass="{base_mass}" x="0" y="0" z="0">
    <origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_box>
</link>

<gazebo reference="kinect_base">
    <material>Gazebo/black</material>
</gazebo>

<joint name="piller2_base_joint" type="fixed">
    <parent link="piller2_link"/>
    <child link="kinect_base"/>
    <origin xyz="0 0 ${piller2_length/2}" rpy="0 0 0"/>
</joint>

<link name="piller2_link">
    <visual>
        <geometry>
            <cylinder length="{piller2_length}" radius="{piller2_radius}"/>
        </geometry>
        <material name="blue"/>
    </visual>
    <collision>
        <origin xyz="0 0 0"/>
        <geometry>
            <cylinder length="{piller2_length}" radius="{piller2_radius}"/>
        </geometry>
    </collision>
</link>

<joint name="piller2_chassis_joint" type="fixed">
    <parent link="chassis"/>
    <child link="piller2_link"/>
    <origin xyz="0.1 0 0.04" rpy="0 0 0"/>
</joint>

```

```

<joint name="kinect_depth_joint" type="fixed">
  <origin xyz="0 0.011 0" rpy="0 0 0" />
  <parent link="camera_link" />
  <child link="camera_depth_frame" />
</joint>

<link name="camera_depth_frame">
  <inertial>
    <mass value="0.01" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.001" ixy="0.0" ixz="0.0"
      iyy="0.001" iyz="0.0"
      izz="0.001" />
  </inertial>
</link>

<joint name="camera_depth_optical_joint" type="fixed">
  <origin xyz="0 0 0" rpy="{-pi/2} 0 {-pi/2}" />
  <parent link="camera_depth_frame" />
  <child link="camera_depth_optical_frame" />
</joint>

<link name="camera_depth_optical_frame">
  <inertial>
    <mass value="0.001" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
      iyy="0.0001" iyz="0.0"
      izz="0.0001" />
  </inertial>
</link>

<joint name="camera_rgb_joint" type="fixed">
  <origin xyz="0 -0.012 0" rpy="0 0 0" />
  <parent link="camera_link" />
  <child link="camera_rgb_frame" />
</joint>

```

```

<link name="camera_rgb_frame">
  <inertial>
    <mass value="0.001" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
      iyy="0.0001" iyz="0.0"
      izz="0.0001" />
  </inertial>
</link>

<joint name="camera_rgb_optical_joint" type="fixed">
  <origin xyz="0 0 0" rpy="{-pi/2} 0 {-pi/2}" />
  <parent link="camera_rgb_frame" />
  <child link="camera_rgb_optical_frame" />
</joint>

<link name="camera_rgb_optical_frame">
  <inertial>
    <mass value="0.001" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
      iyy="0.0001" iyz="0.0"
      izz="0.0001" />
  </inertial>
</link>

</robot>

```

*car\_description/urdf/rplidar.xacro:*

```

<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" >

  <xacro:include filename="$(find car_description)/urdf/inertial_macros.xacro"/>
  <xacro:include filename="$(find car_description)/urdf/materials.xacro"/>

  <xacro:property name="base_length" value="0.1"/>
  <xacro:property name="base_width" value="0.1"/>

```

```

<xacro:property name="base_height" value="0.01"/>
<xacro:property name="base_mass" value="0.005"/>

<xacro:property name="pillar_length" value="0.15"/>
<xacro:property name="pillar_radius" value="0.02"/>

<link name="rplidar_link">
  <visual>
    <origin xyz="0 0 0" rpy="{pi/2} 0 0"/>
    <geometry>
      <mesh filename="package://car_description/meshes/sensors/rplidar.STL" scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://car_description/meshes/sensors/rplidar.STL" scale="0.001 0.001 0.001"/>
    </geometry>
  </collision>
</link>

<joint name="laser_joint" type="fixed">
  <parent link="rplidar_link"/>
  <child link="laser"/>
  <origin xyz="0 0 0.05" rpy="0 0 -{pi/2}"/>
</joint>

<link name="laser">

</link>

<joint name="rplidar_to_base_joint" type="fixed">
  <parent link="rplidar_base"/>
  <child link="rplidar_link"/>
  <origin xyz="-0.04 0.03 {base_height/2}" rpy="0 0 0"/>
</joint>

<link name="rplidar_base">

```

```

<visual>
  <origin xyz="0 0 0"/>
  <geometry>
    <box size="${base_length} ${base_width} ${base_height}"/>
  </geometry>
  <material name="orange"/>
</visual>

<collision>
  <origin xyz="0 0 0"/>
  <geometry>
    <box size="${base_length} ${base_width} ${base_height}"/>
  </geometry>
</collision>

<xacro:inertial_box mass="${base_mass}" x="0" y="0" z="0">
  <origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_box>
</link>

<gazebo reference="rplidar_base">
  <material>Gazebo/black</material>
</gazebo>

<joint name="piller_base_joint" type="fixed">
  <parent link="piller_link"/>
  <child link="rplidar_base"/>
  <origin xyz="0 0 ${piller_length/2}" rpy="0 0 0"/>
</joint>

<link name="piller_link">
  <visual>
    <geometry>
      <cylinder length="${piller_length}" radius="${piller_radius}"/>
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <origin xyz="0 0 0"/>
    <geometry>

```



```

        <cylinder length="{piller_length}" radius="{piller_radius}"/>
    </geometry>
</collision>
</link>

<joint name="piller_chassis_joint" type="fixed">
    <parent link="chassis"/>
    <child link="piller_link"/>
    <origin xyz="-0.1 0 0.10" rpy="0 0 0"/>
</joint>

<gazebo reference="rplidar_link">
    <material>Gazebo/Red</material>

    <sensor name="laser" type="ray">
        <pose> 0 0 0 0 0 0 </pose>
        <visualize>true</visualize>
        <update_rate>10</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>360</samples>
                    <min_angle>-3.14</min_angle>
                    <max_angle>3.14</max_angle>
                </horizontal>
            </scan>
            <range>
                <min>0.3</min>
                <max>12</max>
            </range>
        </ray>
        <plugin name="laser_controller" filename="libgazebo_ros_ray_sensor.so">
            <ros>
                <argument>~/out:=scan</argument>
            </ros>
            <output_type>sensor_msgs/LaserScan</output_type>
            <frame_name>laser_frame</frame_name>
        </plugin>
    </sensor>
</gazebo>

```

```
    </plugin>  
  </sensor>  
</gazebo>  
  
</robot>
```

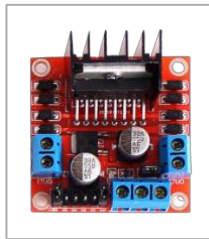
## Appendix C      Data sheets



[Raspberry Pi 4B](#)



[Arduino UNO](#)



[L298N Motor Driver](#)



[RPLIDAR A1](#)



[Xbox 360 Kinect camera](#)