AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
International Credit Hours Engineering Program
(I. CHEP) Design And Production Department

Design of Mechatronic Systems (2) - MCT332

FINAL SUBMISSION

Submitted by:

Name	ID
Fady Sameeh	20P6304
Hossam Torky	20P8211
Abdullah Omar	20P5573
Youssef Ahmed	20P5913
Yehia Khaled	20P6337
Youssof Tamer	20P2497

Supervised by: Dr. Shady Ahmed Maged

ABSTRACT

In response to the evolving landscape of modern military operations, the development of advanced technologies plays a pivotal role in enhancing security measures and situational awareness. This technical report presents the design, development, and implementation of an autonomous military robot specifically engineered for room scanning and intruder elimination. Leveraging cutting-edge robotics, artificial intelligence, and sensing technologies, the system autonomously navigates enclosed spaces, generates accurate 2D maps, and responds effectively to unauthorized intrusions in real-time. The report provides a comprehensive overview of the system architecture, including hardware specifications, software algorithms, and integration methodologies. Furthermore, it outlines the testing protocols, validation procedures, and performance evaluations conducted to ensure the system's reliability, accuracy, and safety in operational scenarios. By amalgamating state-of-the-art technologies, this project contributes to the advancement of military capabilities in reconnaissance and security missions, offering enhanced protection for personnel and assets. The abstract encapsulates the project's objectives, methodologies, and significance, providing a succinct overview of the technical advancements achieved in autonomous military robotics.

Table of Contents

1.0. INTRODUCTION 5
2.0. CAD MODEL 6
2.1. Renders 6
2.2. Sensors & Cameras 7
2.3. Actuators 8
2.4. Mechanical Consideration & Fixation 9
2.4.1. Metal chassis 9
2.4.2. Front wheel assembly 9
2.4.3. Rear wheel assembly 9
3.0. STRESS ANALYSIS 10
3.1. Von Mises Stress 10
3.2. Total Displacement 10
4.0. ELECTRIC COMPONENTS 11
4.1. Raspberry Pi 4 11
4.2. Arduino Mega 11
4.3. Arduino Uno 11
4.4. Motor driver (Cytron) 11
4.5. Battery 12
4.6. Power Circuit 12
4.7. Kill Switch 12
4.8. On/Off Switch 12
5.0. PCB Design 13
6.0. ACTUATOR SIZING 14
6.1. Using Software 14
6.2. Using Manual Calculations 17
7.0. FLOWCHARTS 19
7.1. Explanation 19
7.2. Charts 20
8.0. SIMULATION 22
9.0. COMPONENTS & PRICES 24
9.1. Mechanical Components 24
9.2. Electrical Components 25
10.0. REFERENCES 58

Table of Figures

Figure 1 Special Forces Attack 5
Figure 2 Full Render View 5 6
Figure 3 Full Render View 4 6
Figure 4 Full Render View 3 6
Figure 5 Full Render View 2 6
Figure 6 Full Render View1 6
Figure 7 Lidar 7
Figure 8 Ultrasonic sensor 7
Figure 9 Motor Encoder 7
Figure 10 Camera 7
Figure 11 DC Motor 8
Figure 12 DC motor 2 8
Figure 13 Servo Motor 8
Figure 14 Metal Chassis 9
Figure 15 Front Wheel 9
Figure 16 Rear wheel 9
Figure 17 Bearing Mount 9
Figure 18 Bearing Fixation Illustration 9
Figure 19 Von Mises Stress 10
Figure 20 Total Displacement stress 10
Figure 21 Raspberry Pi 11
Figure 22 Arduino Mega 11
Figure 23 Self made Arduino Uno 11
Figure 24 Cytron 11
Figure 25 Battery 12
Figure 26 Power Circuit 12
Figure 27 Kill Switch 12
Figure 28 On/Off Switch 12
Figure 29 PCB Design 13
Figure 30 PCB circuit 13
Figure 31 Left Front Wheel Contact with Ground 14
Figure 32Right Front wheel Contact w/ ground 14
Figure 33 Left Rear Wheel Contact with Ground 14
Figure 34 Right Rear Wheel Contact with Ground 14
Figure 35 Define Motor for Left Front Wheel 15
Figure 36 Define Motor for Left Front Wheel 15
Figure 37 Trapezoidal Motor Motion Profile 15
Figure 38 Gravity Direction 15
Figure 39 Setting Results and Plots Options 16
Figure 40 RMS Torque Sensor 16
Figure 41 Calculations 1 17
Figure 42 Calculations 2 18
Figure 43 Flow chart 1 20
Figure 44 Flow chart 2 20
Figure 45 Flow chart 3 21

1.0. INTRODUCTION

In modern military operations, the need for advanced surveillance and defense mechanisms has become increasingly paramount. The evolution of robotics and artificial intelligence presents an opportunity to enhance situational awareness and security measures. In response to this demand, this technical report outlines the design, development, and implementation of an autonomous military robot tasked with the dual objectives of room scanning and intruder elimination.

Figure 1 Special Forces Attack

The primary objective of this project is to create a robust and efficient system capable of autonomously scanning enclosed spaces, generating accurate 2D maps of the environment, and responding effectively to the presence of unauthorized individuals. The integration of cutting-edge technologies such as sensors, computer vision, and machine learning algorithms enables the robot to navigate complex environments, identify potential threats, and take appropriate actions in real-time.

This report will provide a comprehensive overview of the various components and subsystems incorporated into the robot's design, including hardware specifications, software architecture, and algorithmic implementations. Additionally, it will detail the methodologies employed for testing, validation, and performance evaluation to ensure the system's reliability, accuracy, and safety in operational scenarios.

By combining state-of-the-art robotics with advanced AI capabilities, this project aims to contribute to the enhancement of military capabilities in reconnaissance and security missions. The development of an autonomous robot capable of seamlessly scanning rooms and neutralizing intruders represents a significant advancement in defense technology, offering enhanced protection for personnel and assets in diverse operational environments.

Throughout this report, key design considerations, technical challenges, and future directions for improvement will be addressed, providing insights into the development process and potential avenues for further research and development in the field of autonomous military robotics.

2.0. CAD MODEL

2.1. Renders

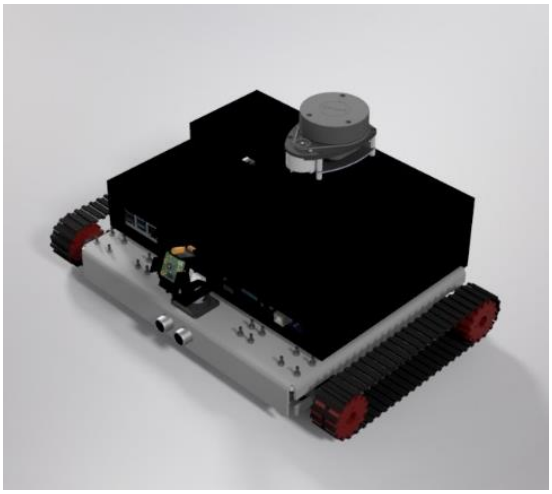


Figure 6 Full Render View1

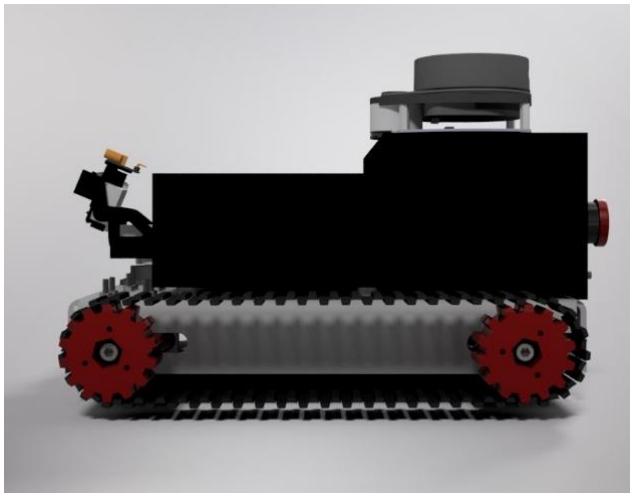


Figure 5 Full Render View 2

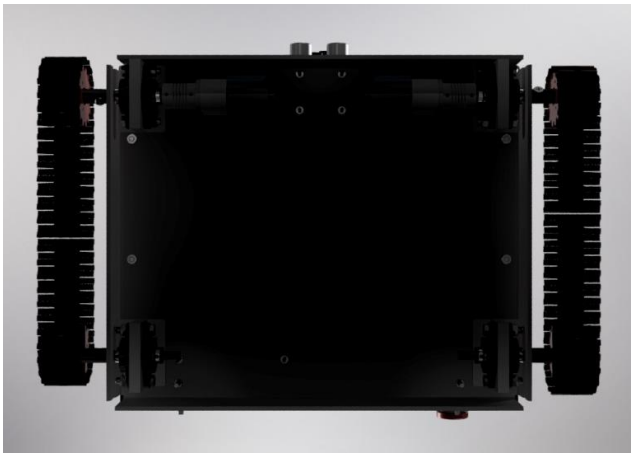


Figure 4 Full Render View 3

Figure 3 Full Render View 4

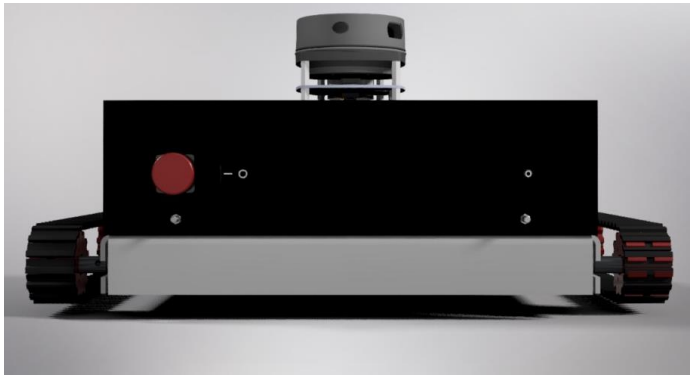


Figure 2 Full Render View 5

2.2. Sensors & Cameras

1. Lidar is a remote sensing technology that uses lasers to measure distances and create precise 2D maps of the surrounding environment.

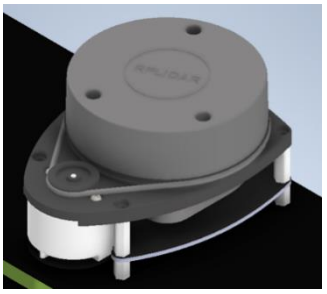


Figure 7 Lidar

2. Ultrasonic sensors are used for distance measurement and object detection. They work by emitting high-frequency sound waves and measuring the time it takes for the waves to bounce back after hitting an object. This information is then used to calculate the distance to the object.

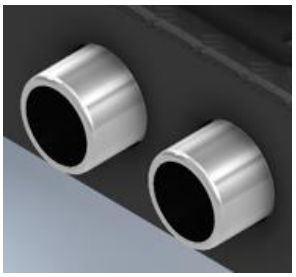


Figure 8
Ultrasonic sensor

3. A motor encoder is a device used to measure the position, speed, and direction of a motor shaft. It consists of a sensor that detects the rotation of the motor shaft and provides feedback in the form of electrical signals.

Figure 9 Motor
Encoder

4. A camera in a robot serves as an essential sensory component, enabling visual perception and recognition of the surrounding environment. It captures images or video footage that can be processed and analyzed by the robot's algorithms. The camera provides valuable information for tasks such as object detection, recognition, tracking, navigation, and mapping.

Figure 10
Camera

2.3. Actuators

1. A DC motor is responsible for the motion of the Lidar while rotating.

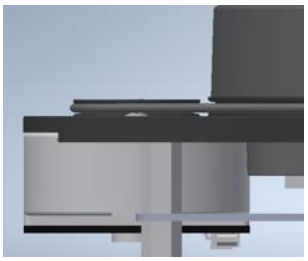


Figure 11 DC Motor

2. A DC motor and its encoder is responsible for generating and controlling the wheels' motion which will move the whole robot.

Figure 12 DC motor 2

3. Servo motor is used to control the motion of the camera

Figure 13 Servo Motor

2.4. Mechanical Consideration & Fixation

2.4.1. Metal chassis

The chassis design is brilliant as it has small thickness to provide a light weighted chassis.

Also we chose to use a metal chassis to add rigidity, help the chassis overcome heavier loads and unfortunate circumstances.

Figure 14 Metal Chassis

2.4.2. Front wheel assembly

The motor is mounted to the floor using a custom holder. Its shaft, which holds a bearing, is connected to an extension rod that links to a hexagonal coupler fitted directly into the wheel.

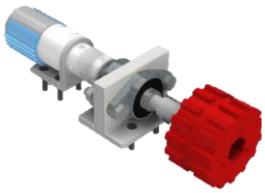


Figure 15 Front Wheel

2.4.3. Rear wheel assembly

Using a custom holder. Its shaft, which holds a bearing, is connected to an extension rod that links to a hexagonal coupler fitted directly into the wheel.

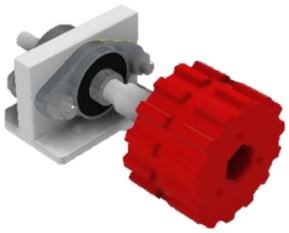


Figure 16 Rear wheel

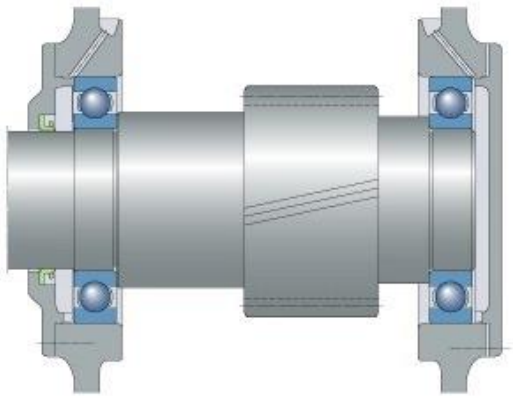


Figure 18 Bearing Fixation Illustration

Figure 17 Bearing Mount

3.0. STRESS ANALYSIS

3.1. Von Mises Stress

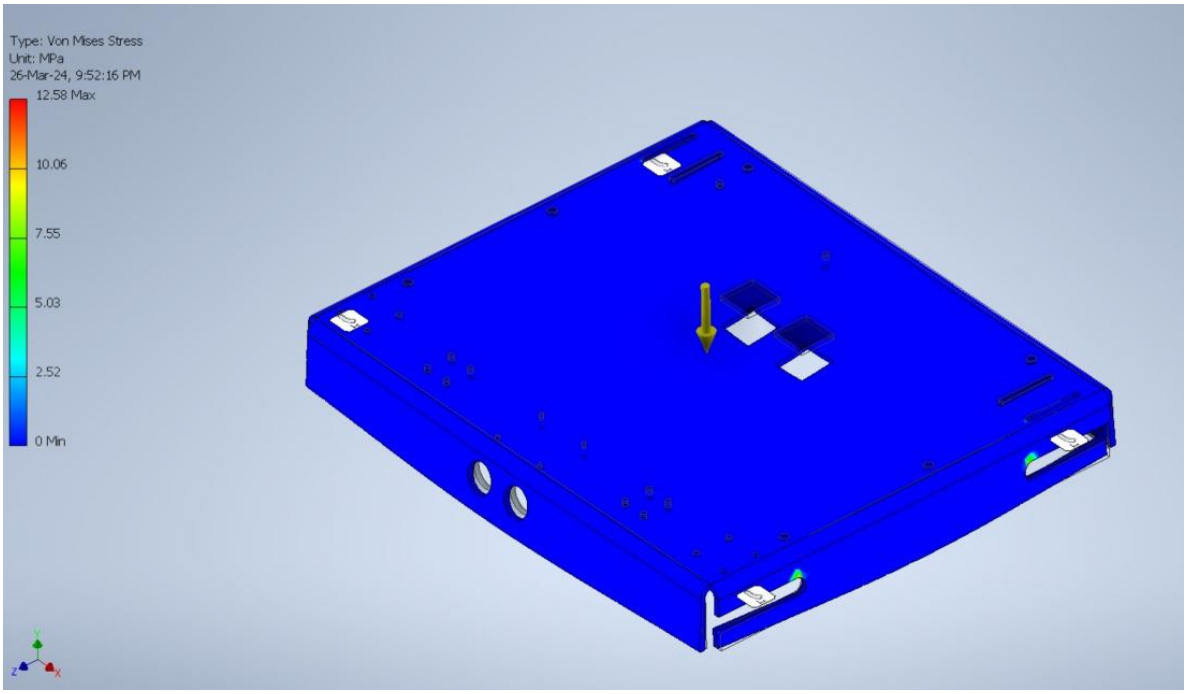


Figure 19 Von Mises Stress

3.2. Total Displacement

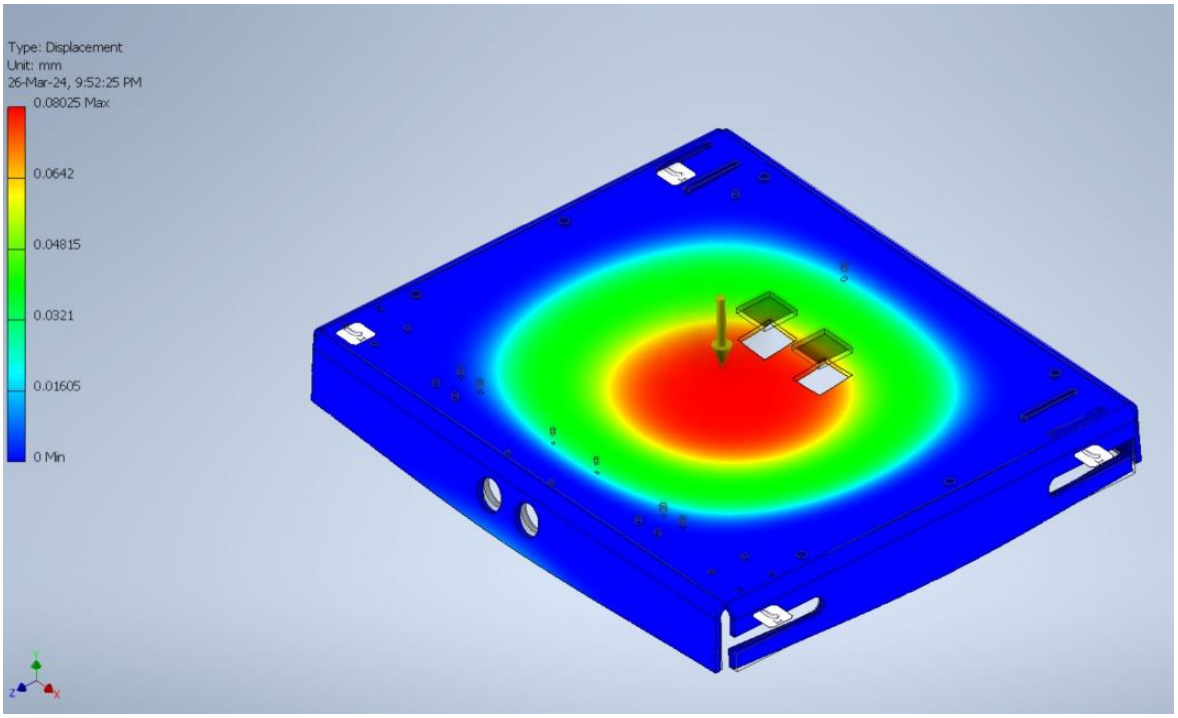


Figure 20 Total Displacement stress

4.0. ELECTRIC COMPONENTS

4.1. Raspberry Pi 4

A small, affordable single-board computer with GPIO pins, running a Linux-based OS, suitable for a wide range of projects from basic computing to IoT applications.

Figure 21 Raspberry Pi

4.2. Arduino Mega

An enhanced version of Arduino Uno with a larger number of input/output pins, suitable for complex projects requiring more connections.

Figure 22 Arduino Mega

4.3. Arduino Uno

A versatile microcontroller board with digital and analog input/output pins, ideal for prototyping and DIY projects.

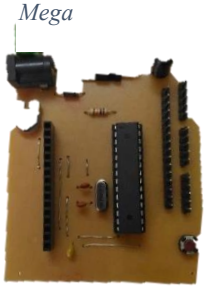


Figure 23 Self made Arduino Uno

4.4. Motor driver (Cytron)

An electronic module that interfaces between a microcontroller (like Arduino) and electric motors, enabling precise control over speed and direction.

Figure 24 Cytron

4.5. Battery

Rechargeable Battery 10000mA.

Figure 25 Battery

4.6. Power Circuit

Responsible for power distribution to all electric components and boards.

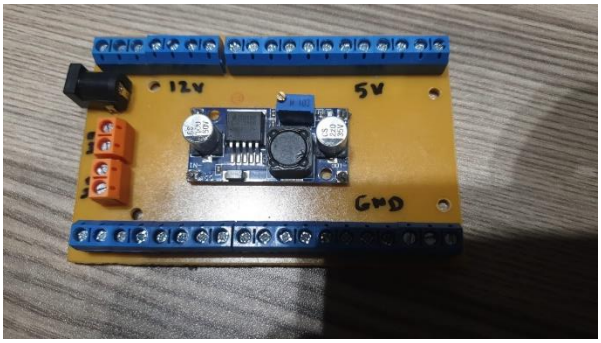


Figure 26 Power Circuit

4.7. Kill Switch

Emergency stopping the robot.

Figure 27 Kill Switch

4.8. On/Off Switch

Figure 28 On/Off Switch

5.0. PCB Design

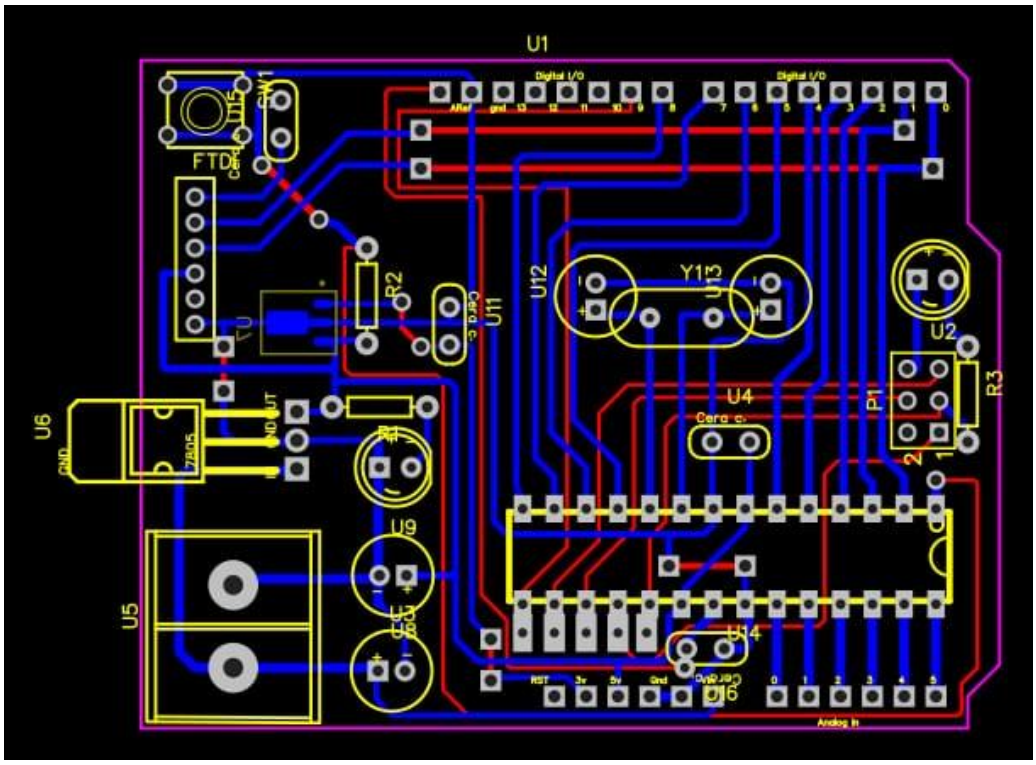


Figure 29 PCB Design

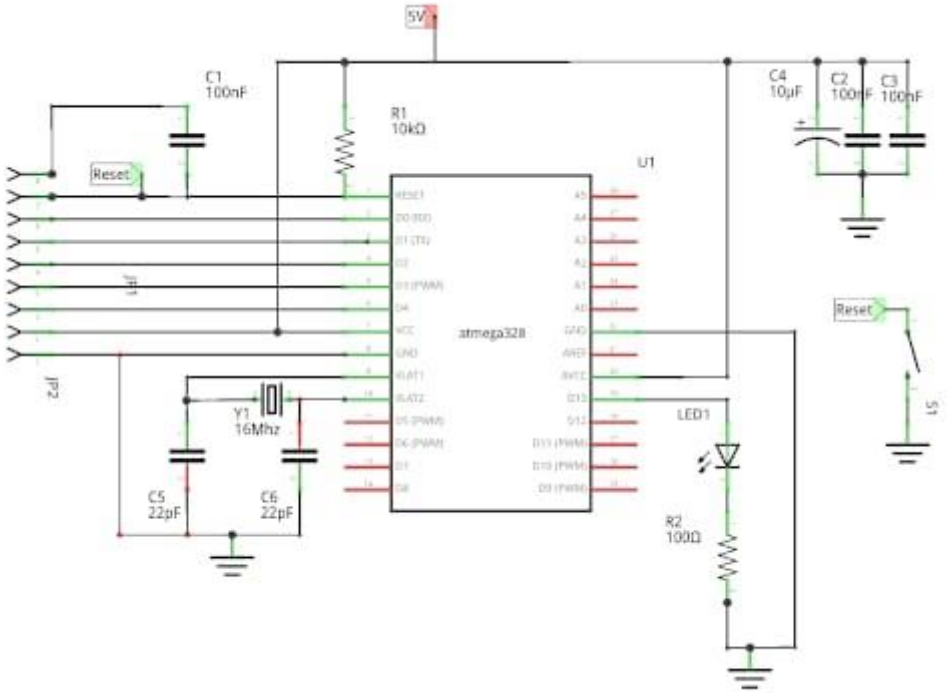


Figure 30 PCB circuit

6.0. ACTUATOR SIZING

6.1. Using Software

1. Define contact between the wheels and the ground plate

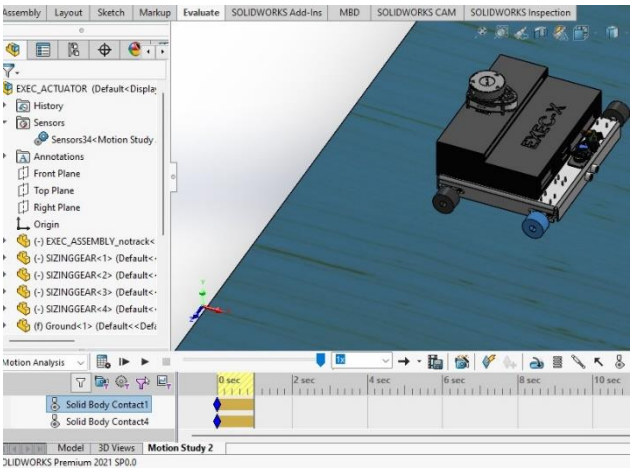


Figure 32Right Front wheel Contact w/ ground

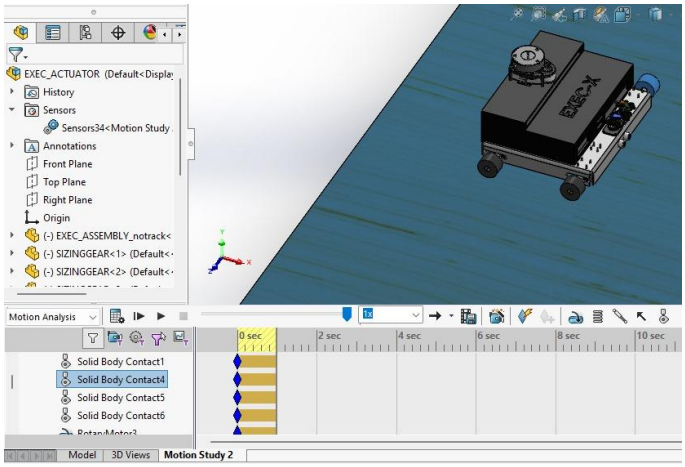


Figure 31 Left Front Wheel Contact with Ground

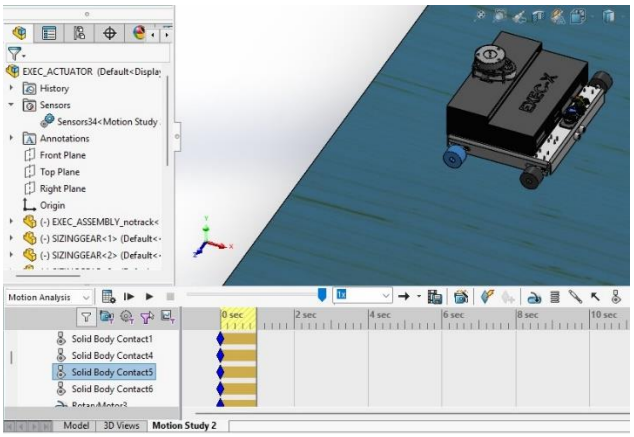


Figure 34 Right Rear Wheel Contact with Ground

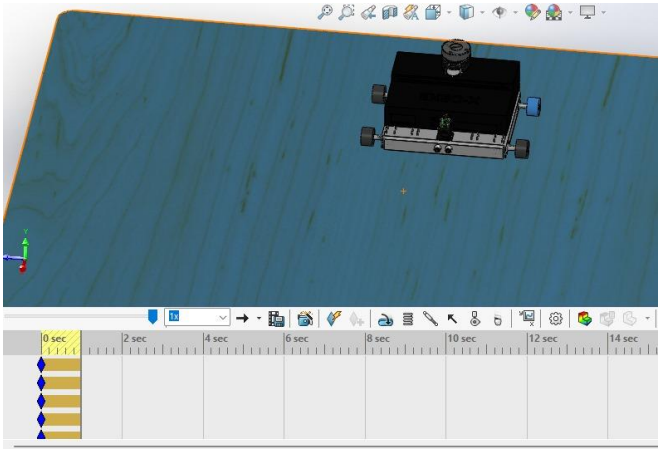


Figure 33 Left Rear Wheel Contact with Ground

2. Define existence of 2 motors driving front wheels as follows, select rotary motor by default, select cylindrical face on the wheel as the first selection, select the robot body as the third selection to be the component that the wheel moves relative to.

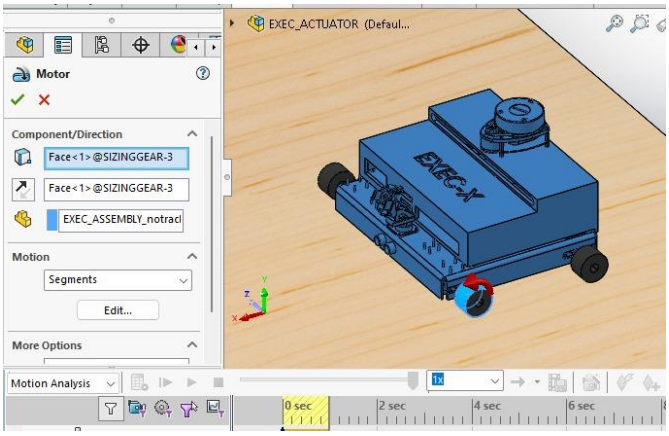


Figure 35 Define Motor for Left Front Wheel

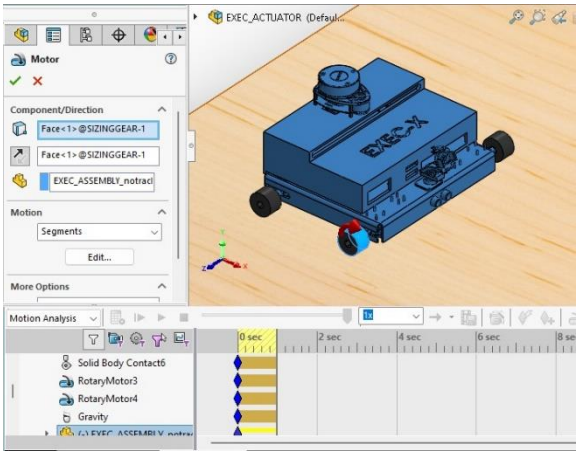


Figure 36 Define Motor for Left Front Wheel

3. Motor desired motion profile can be set from the motion section, for this simulation we will set it as trapezoidal motion profile.

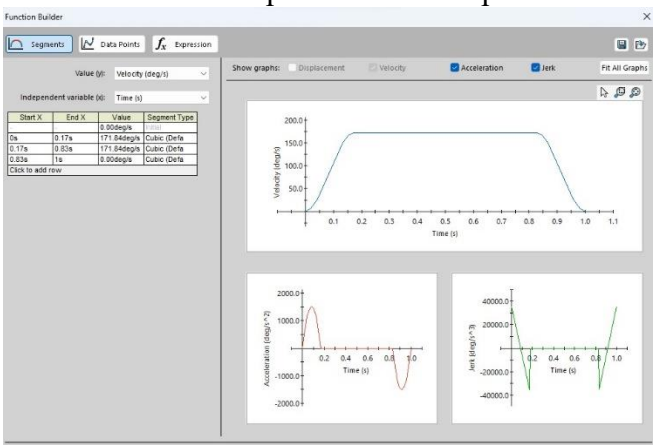


Figure 37 Trapezoidal Motor Motion Profile

4. Adjust the gravity direction to be in negative Y direction

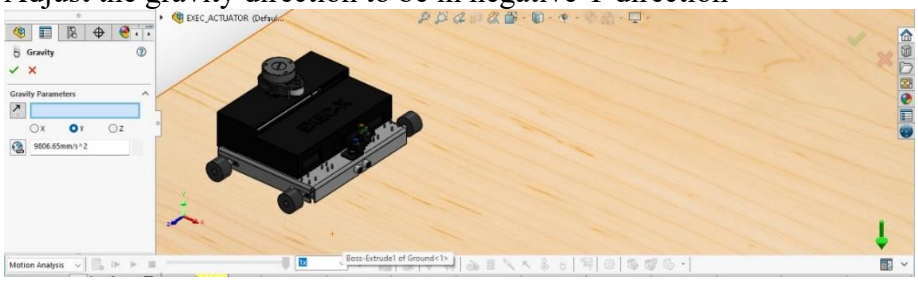


Figure 38 Gravity Direction

5. To view motor torque, change “Basic motion” to “motion analysis”, then click on plots and results and choose the following option

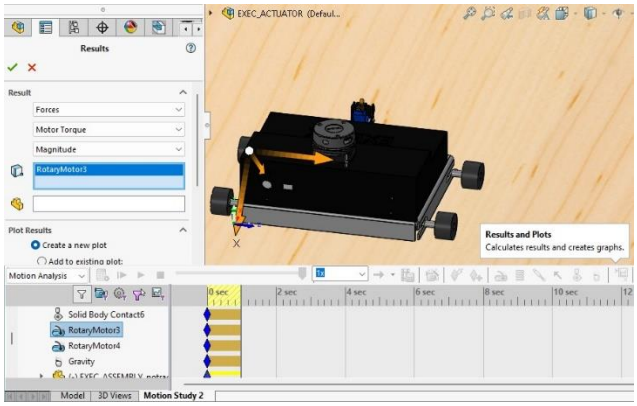


Figure 39 Setting Results and Plots Options

6. Create a sensor to evaluate the RMS torque, from evaluate tab choose sensors, then set the following options

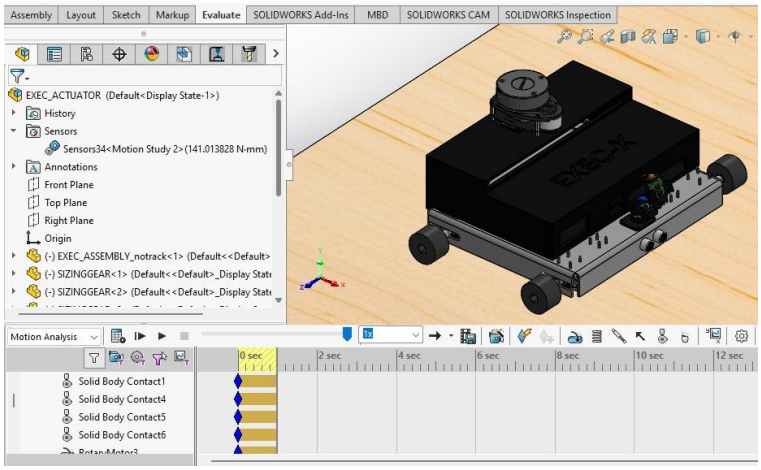


Figure 40 RMS Torque Sensor

6.2. Using Manual Calculations

Actuator Sizing

$F_a = \frac{1}{2} \times C_d \times \rho \times A_p \times V^2$

$= \frac{1}{2} \times 0.1 \times 1.2 \times (0.4 \times 0.28) \times 0.1^2$

$= 6.72 \times 10^{-5}$ ~ negligable

$F_{rolling} = \mu_r \times mg = 0.15 \times 5 \times 9.81 = 7.3575$

$\therefore T_R = 7.3575 \times 0.04 = 0.2943$

$T_m = T_R + J\ddot{\theta}$

$J_{Tot.} = J_m + J_{eff.wheel} + J_{mass} + J_{belt}$

$J_{mass} = \frac{m \times r^2}{\eta \times N^2} = \frac{5 \times 40^2}{0.9 \times 1^2} = 8888.89 \ Kg.mm^2$

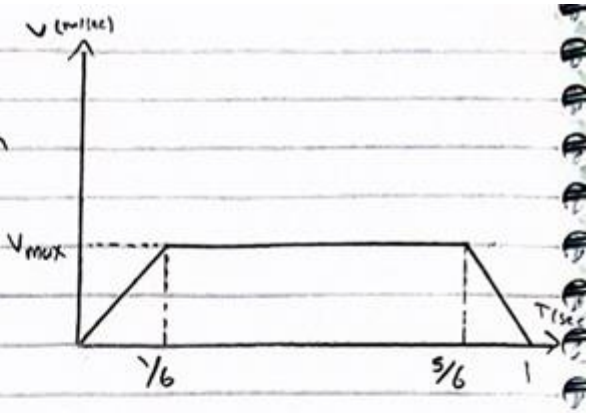
$J_{belt} = mr^2 = 0.065 \times 40^2 = 104 \ Kg.mm^2$

$J_m = 888.89 \ Kg.mm^2$

$J_{eff.wheel} = \frac{1}{2} mr^2 = \frac{1}{2} \times 0.025 \times 40^2 = 20 \ Kg.mm^2$

Figure 41 Calculations 1

$d = 10$ cm , Time = 1 sec.



$2 \times \frac{1}{2} \times V_{max} \times \frac{1}{6} + V_{max} \times \frac{2}{3} = 10$ cm

$\therefore V_{max} = 12$ cm/sec

$\omega_{max} = \frac{120}{40} = 3$ rad/sec

$= 28.64$ rpm

$\alpha = \frac{3}{1/6} = 18$ rad/sec^2

$\therefore J_{total} = 8888.89 + 828.89 + 104 + 20 = 9901.78$ kg.mm^2

$\alpha = 18$ rad/sec^2

$\therefore T_{m_{max}} = T_R + J\alpha = 0.2943 + 18 \times (9901.78 \times 10^{-6}) = 0.4725$ Nm

$T_{m_{min}} = T_R - J\alpha = 0.2943 - 18 \times (9901.78 \times 10^{-6}) = 0.11606$ Nm

$$T_{rms} = \sqrt{\frac{0.4725^2 \times \frac{1}{6} + 0.2943^2 \times \frac{2}{3} + 0.11606^2 \times \frac{1}{6}}{1}} = 0.31176 \text{ Nm}$$

For only one motor $= \frac{1}{2} \times 0.31176 = 0.15588$ Nm.

Figure 42 Calculations 2

7.0. FLOWCHARTS

7.1. Explanation

The flowchart illustrates the connectivity of our electrical components, delineating their roles and interactions within the system architecture. Initially, the Raspberry Pi establishes a connection with the LiDAR sensor, pivotal for mapping and localization tasks. This communication occurs via UART, facilitating seamless data exchange. Additionally, the Raspberry Pi interfaces with the Camera v1.3, designated for human detection purposes, utilizing the MIPI camera serial interface protocol. Power is supplied to the Raspberry Pi through an external power bank, ensuring continuous operation.

Furthermore, a custom Arduino Uno board is integrated with an ultrasonic sensor and servo motors. The ultrasonic sensor assumes responsibility for obstacle detection, contributing to the system's navigational capabilities. Simultaneously, the servo motors govern the motion of the camera, enhancing its agility and versatility. An Arduino Mega board is linked with the Citron motor driver, tasked with motor control functions. Notably, the motor incorporates an encoder to capture positional data, transmitted to the Arduino Mega via the I2C communication protocol.

Both the Arduino Uno and Arduino Mega boards are powered by a 10000mA battery source, ensuring sufficient energy for sustained operation. Moreover, they establish communication with the Raspberry Pi via UART, facilitating coordinated actions and data exchange between the components. This systematic arrangement of components and communication protocols ensures the efficient functioning of the system, enabling robust performance and adaptability to diverse operational scenarios.

7.2. Charts

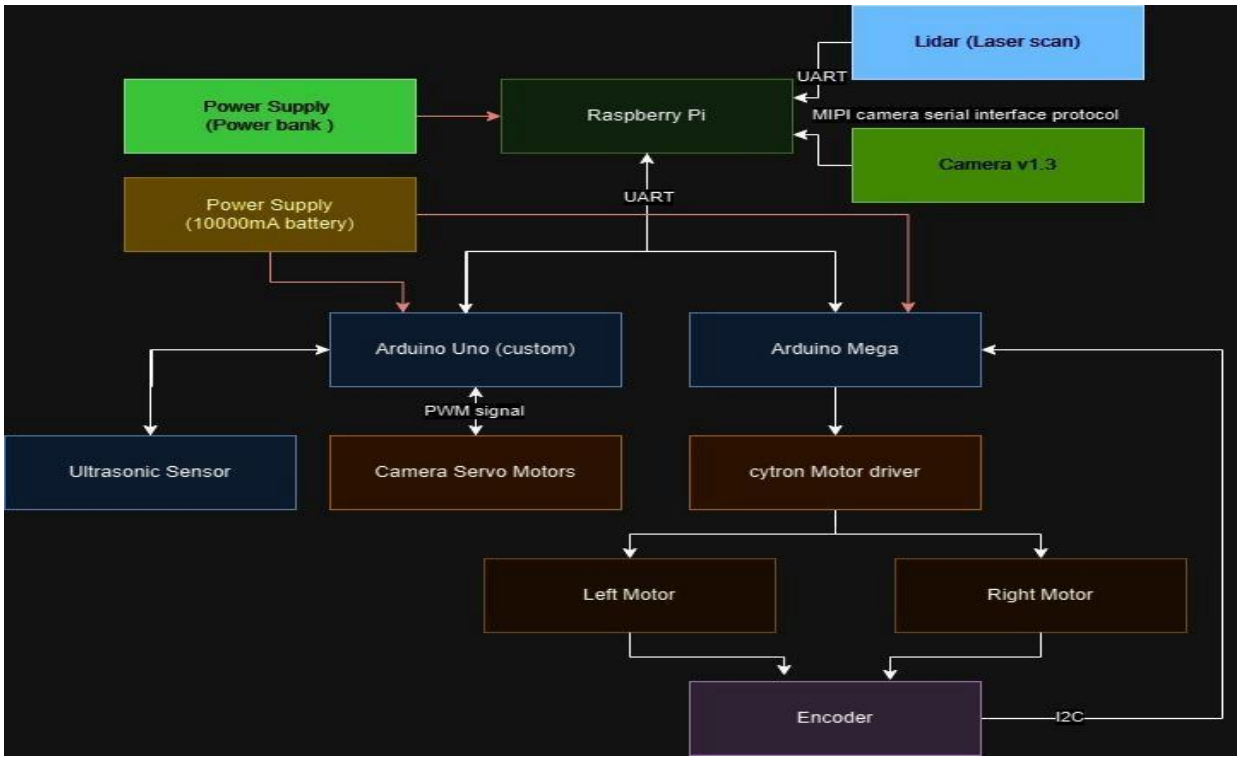


Figure 43 Flow chart 1

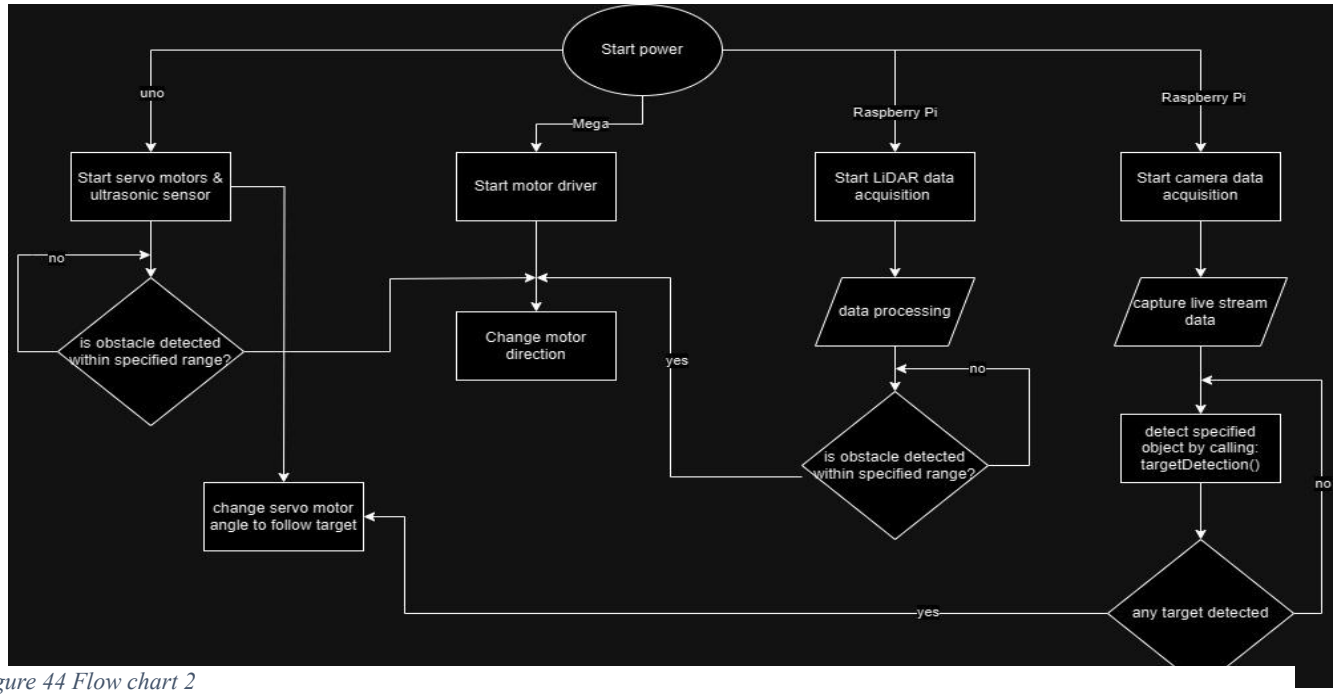


Figure 44 Flow chart 2

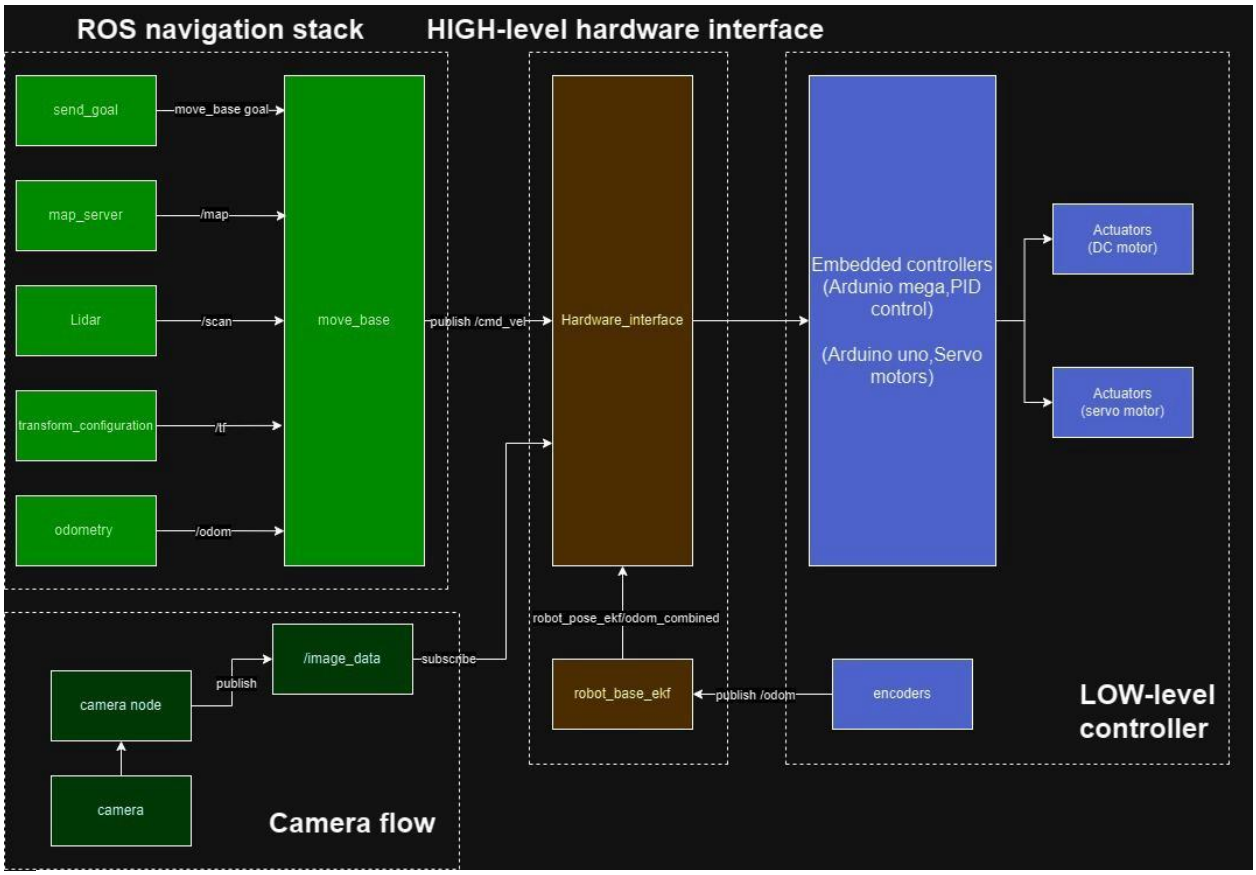
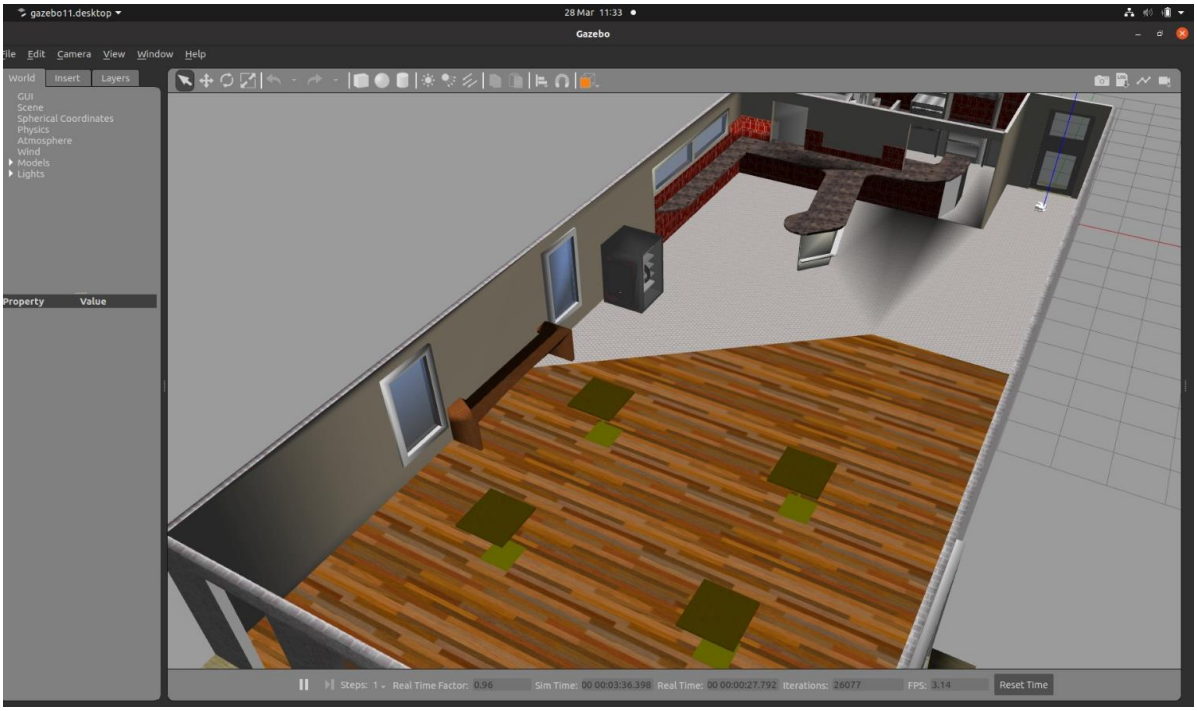


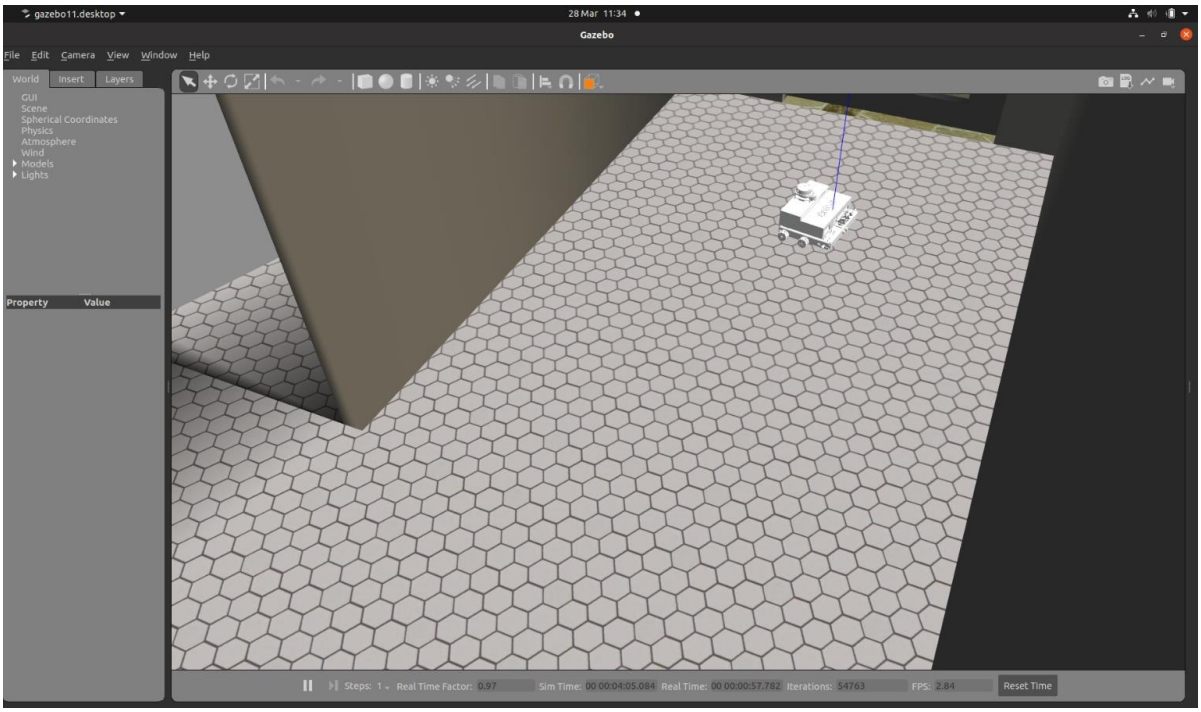
Figure 45 Flow chart 3

8.0. SIMULATION

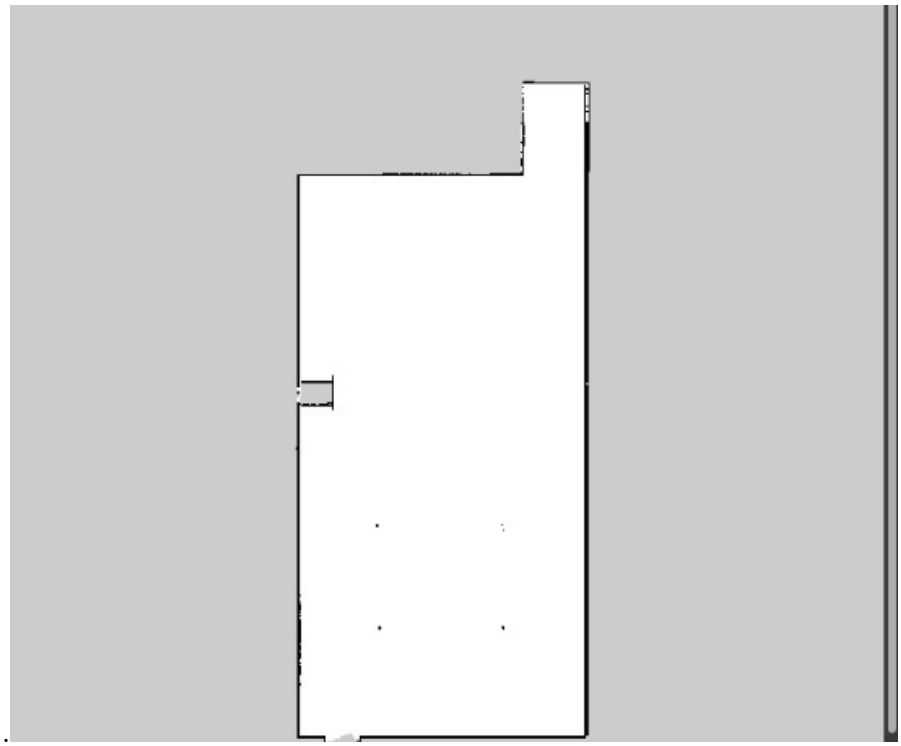
- Map environment.



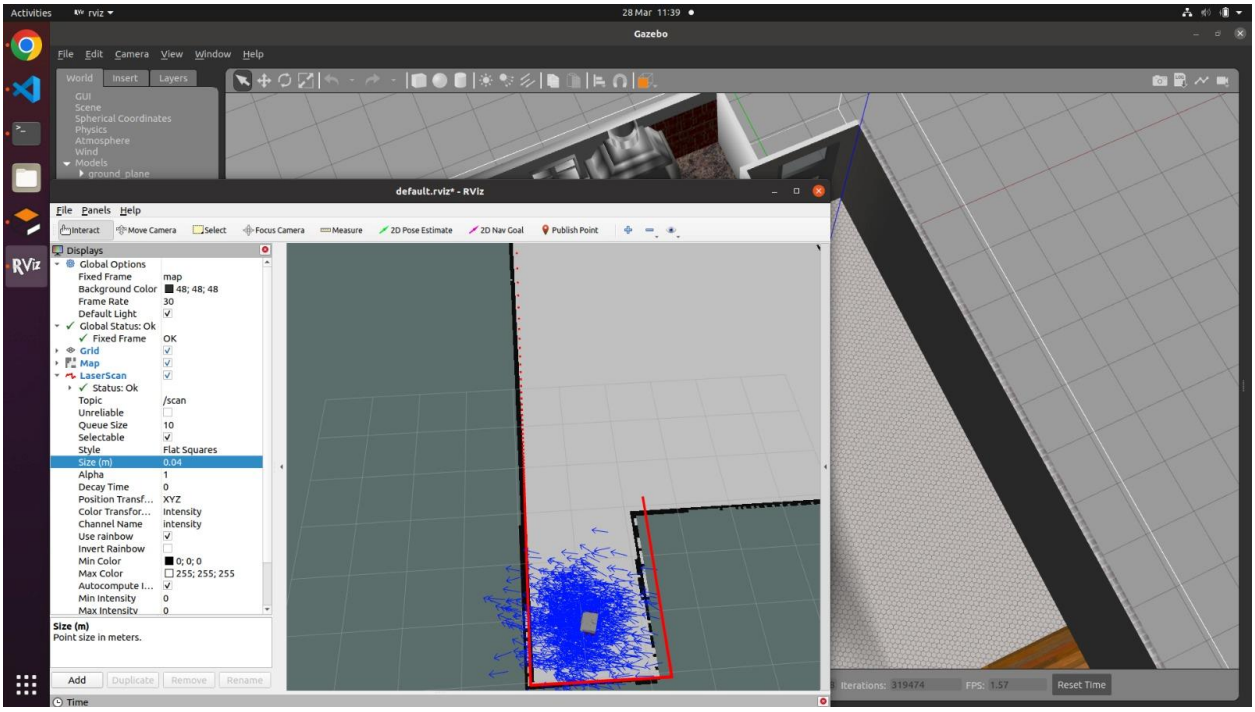
- Robot pose in environment.



- Scanned map using lidar



- Live pose and navigation inside the map.



9.0. COMPONENTS & PRICES

9.1. Mechanical Components

MATERIALS		
NAME	**QUANTITY**	**PRICE**
Aluminum sheet		
Acrylic - black		
PAEK plastic		
Steel-mild		
COMPONENTS		
NAME	**QUANTITY**	**PRICE**
Bearing housing mounts	8	
M5 bolts		
M5 nuts		
M3 bolts		
M3 nuts		
Pan tilt mechanism	1	
Aluminum spacers – Ø5/6mm	4	
Fixation brackets 20x20mm	2	

9.2. Electrical Components

NAME	QUANTITY	PRICE
Raspberry Pi	1	
Arduino mega	1	
Arduino Uno (custom)	1	
Cytron motor driver	1	
Battery 10000mAh	1	
Custom power circuit	1	
Servo motors	2	
DC motor with encoders	2	
Raspberry camera v1.3	1	
RPlidar – Slamtech	1	
Laser Module	1	
Kill switch	1	
On/Off switch	1	
Power bank 12v – 5000mAh	1	
Ultrasonic sensor	1	
Jumper wires	30	

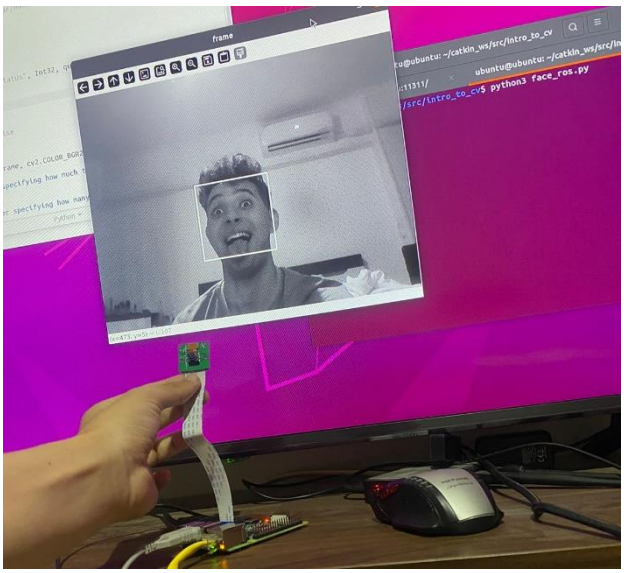
10.0. Camera Visualization & Face Detection

10.1. Code

```
import cv2

import rospy

from std_msgs.msg import Int32

if _name_ == "_main_":

    # Load the face detector model

    face_cascade = cv2.CascadeClassifier('haar/haarcascade_frontalface_default.xml')

    # Load the demo video

    vid_path = "videos/face_demo.mp4"

    cap = cv2.VideoCapture(0)

    # Initialize ROS node

    rospy.init_node("face_detector_node")

    pub = rospy.Publisher("face_detector/status", Int32, queue_size=10)

    while cap.isOpened():

        ret, frame = cap.read()

        # Reset each frame

        face_detected_in_this_frame = False

        if ret:

            image_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # scale_factor: Parameter specifying how much the image size is reduced at each image
scale.

            scale_factor = 1.3

            # min_neighbours: Parameter specifying how many neighbors each candidate rectangle
should have to retain it.

            min_neighbours = 5

            # Perform the faces detection

            faces = face_cascade.detectMultiScale(image_gray, scale_factor, min_neighbours)
```

```
for (x,y,w,h) in faces:
    # Draw a rectangle at the detected location of the face
    cv2.rectangle(image_gray, (x,y), (x+w,y+h), (255, 255, 255) ,2)
    # Set the bool face_detected_in_this_frame to True
    face_detected_in_this_frame = True
# Publish the status of the detector
if face_detected_in_this_frame:
    pub.publish(Int32(1))
else:
    pub.publish(Int32(0))
# Show the frame
cv2.imshow('frame', image_gray)
# Check if the user has pressed ESC key
c = cv2.waitKey(1)
if c == 27:
    cv2.destroyAllWindows()
    break   # exit if ESC is pressed
```

10.2. Hardware Implementation



11.0. PID Control

11.1. Code

```
#include <PID_v1.h>

// Motor Control Pins for Motor 1
const int motor1PWM = 5;
const int motor1Direction = 7;

// Motor Control Pins for Motor 2
const int motor2PWM =44 ;
const int motor2Direction = 45;

// Encoder Pins (if used)
const int encoderPinA1 = 2; // Encoder channel A for motor 1 connected to digital pin 2
const int encoderPinB1 = 3; // Encoder channel B for motor 1 connected to digital pin 3
const int encoderPinA2 = 20; // Encoder channel A for motor 2 connected to digital pin 4
const int encoderPinB2 = 21; // Encoder channel B for motor 2 connected to digital pin 5

// Variables for encoder (if used)
volatile long encoderCount1 = 0; // Encoder count for motor 1
volatile long encoderCount2 = 0; // Encoder count for motor 2
unsigned long prevTime = 0;       // Previous time for measuring speed
float motorSpeed1 = 0;            // Motor 1 speed in revolutions per second
float motorSpeed2 = 0;            // Motor 2 speed in revolutions per second

// PID Parameters
float Setpoint = 0; // Desired speed for both motors
double Input1 = 0;   // Current speed for motor 1
double Input2 = 0;   // Current speed for motor 2
```

```
double Output = 0;   // Common PWM value
double Kp = 2.5;       // Proportional gain
double Ki = 70;       // Integral gain
double Kd = 0;       // Derivative gain

// Low Pass Filter Parameters
const float alpha = 0.2; // Filter coefficient

// Create a single PID object for both motors
PID myPID(&Input1, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void setup() {
  Serial.begin(9600);

  // Prompt user to input desired motor speeds
  Serial.println("Enter desired motor speeds (RPM): ");
  while (!Serial.available()) {
    // Wait for user input
  }
  Setpoint = Serial.parseFloat(); // Read desired speed from serial monitor

  // Set up motor control pins
  pinMode(motor1PWM, OUTPUT);
  pinMode(motor1Direction, OUTPUT);
  pinMode(motor2PWM, OUTPUT);
  pinMode(motor2Direction, OUTPUT);

  // Initialize the direction pins for both motors (rotate in opposite directions)
  digitalWrite(motor1Direction, LOW); // Rotate motor 1 in forward direction
  digitalWrite(motor2Direction, HIGH);  // Rotate motor 2 in reverse direction
```

```
  // Set up encoder pins
  pinMode(encoderPinA1, INPUT_PULLUP);
  pinMode(encoderPinB1, INPUT_PULLUP);
  pinMode(encoderPinA2, INPUT_PULLUP);
  pinMode(encoderPinB2, INPUT_PULLUP);

  // Attach interrupts for encoder readings
  attachInterrupt(digitalPinToInterrupt(encoderPinA1), encoderISR1, RISING);
  attachInterrupt(digitalPinToInterrupt(encoderPinA2), encoderISR2, RISING);

  // Initialize the PID controller
  myPID.SetMode(AUTOMATIC);
  myPID.SetSampleTime(100); // Set the PID sample time in milliseconds
  myPID.SetOutputLimits(0, 255); // Set the output limits for both motors
}

void loop() {
  // Calculate motor speeds (if using encoders)
  unsigned long currentTime = millis();
  if (currentTime - prevTime >= 100) { // Update speeds every 100 milliseconds
    // Calculate speed for motor 1
    motorSpeed1 = (float(encoderCount1) / 550.0) * 1000.0 / (currentTime - prevTime); // Calculate speed
in revolutions per second
    encoderCount1 = 0; // Reset encoder count for motor 1

    // Calculate speed for motor 2
    motorSpeed2 = (float(encoderCount2) / 550.0) * 1000.0 / (currentTime - prevTime); // Calculate speed
in revolutions per second
    encoderCount2 = 0; // Reset encoder count for motor 2
```

```
  prevTime = currentTime; // Update previous time
 }

 // Apply low pass filter to smooth out noise
 static float filteredSpeed1 = 0;
 static float filteredSpeed2 = 0;
 filteredSpeed1 = (1 - alpha) * filteredSpeed1 + alpha * motorSpeed1;
 Input1 = filteredSpeed1;
 filteredSpeed2 = (1 - alpha) * filteredSpeed2 + alpha * motorSpeed2;
 Input2 = filteredSpeed2;

 // Compute the PID output
 myPID.Compute();

 // Apply the output to control the motor speeds
 analogWrite(motor1PWM, Output);
 analogWrite(motor2PWM, Output);



 Serial.print(Setpoint);
 Serial.print(" ");
 Serial.print(motorSpeed1);
 Serial.print(" ");
 Serial.print(motorSpeed2);
 Serial.println();
 delay(1);
}

// Interrupt Service Routine (ISR) for encoder (if used)
void encoderISR1() {
```

```
  if (digitalRead(encoderPinB1) == HIGH) {
    encoderCount1--;
  } else {
    encoderCount1++;
  }
}

// Interrupt Service Routine (ISR) for encoder (if used)
void encoderISR2() {
  if (digitalRead(encoderPinB2) == HIGH) {
    encoderCount2++;
  } else {
    encoderCount2--;
  }
}
```

Explanation Of Code :

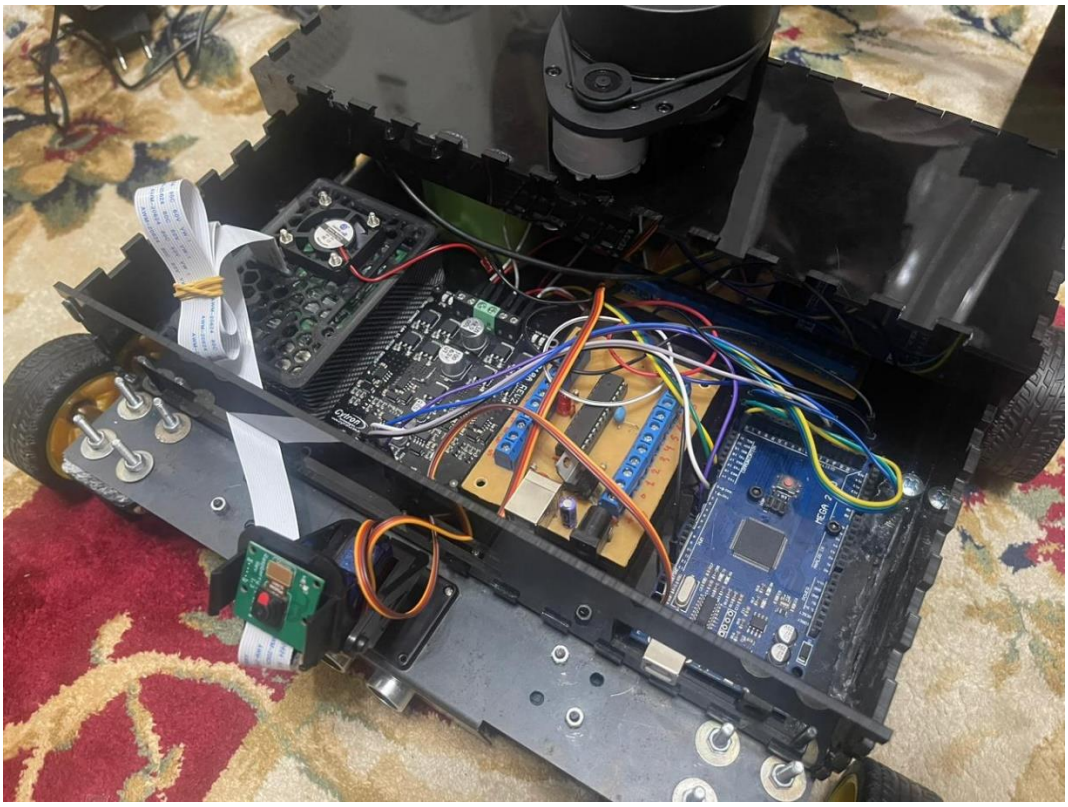
This code sets up two motors, reads their speeds using encoders, filters the speed data, and applies PID control to maintain a set speed for both motors. Here's a breakdown of the key parts of your code:

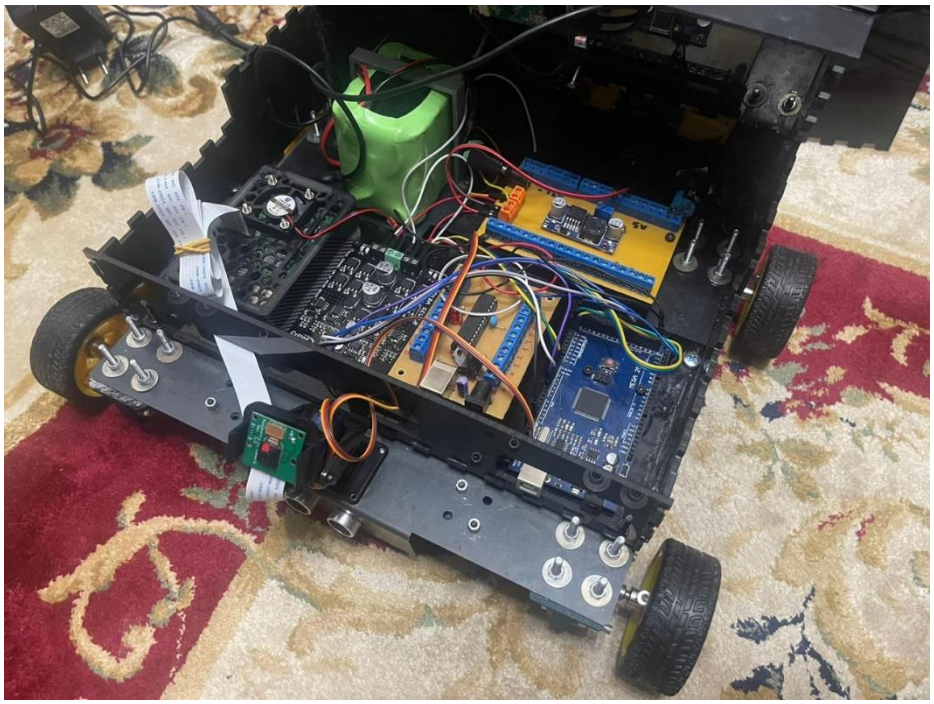
1. **Motor Control Pins:** Define the pins for controlling the motors (PWM for speed control and direction pins).
2. **Encoder Pins:** Define the pins for reading encoder signals from the motors to measure their speed.
3. **Variables:** Define variables for encoder counts, motor speeds, PID parameters, and filter coefficient.
4. **PID Object:** Create a PID object for controlling the motors based on the encoder feedback.
5. **Setup Function:** Set up the serial communication, motor control pins, encoder pins, interrupts for encoder readings, and initialize the PID controller.

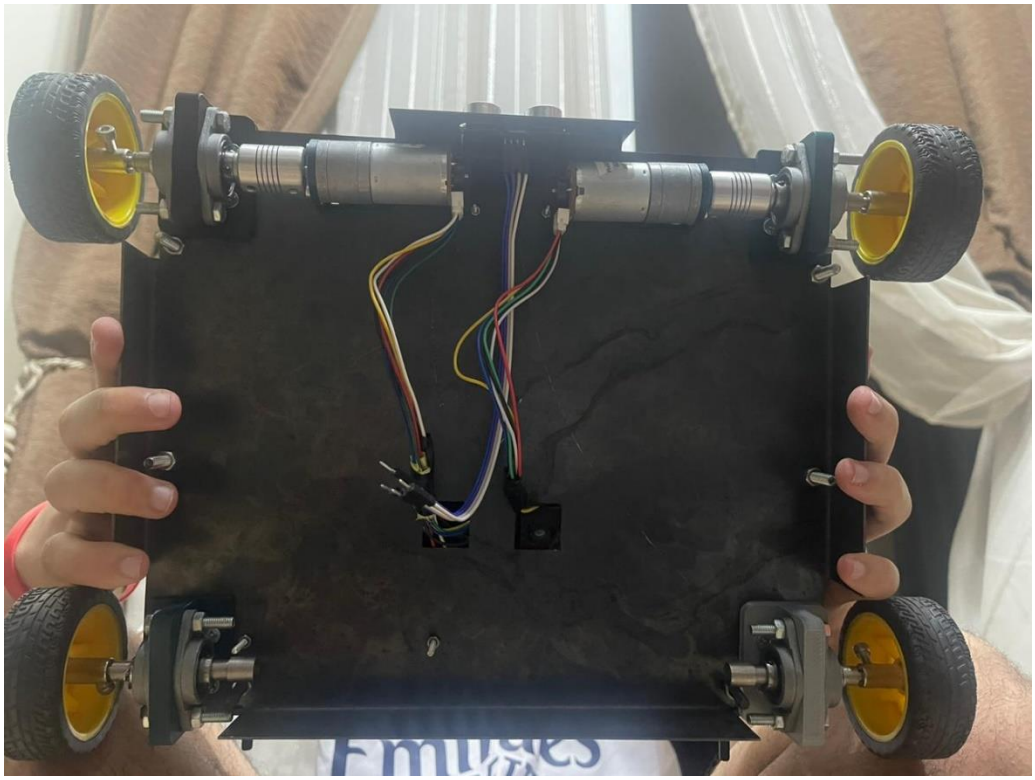
6. **Loop Function:** Continuously read the encoder counts, calculate motor speeds, apply a low pass filter to smooth out noise, compute the PID output, and control the motor speeds using PWM.
7. **Encoder Interrupt Service Routines (ISRs):** Increment or decrement the encoder counts based on the encoder signals.

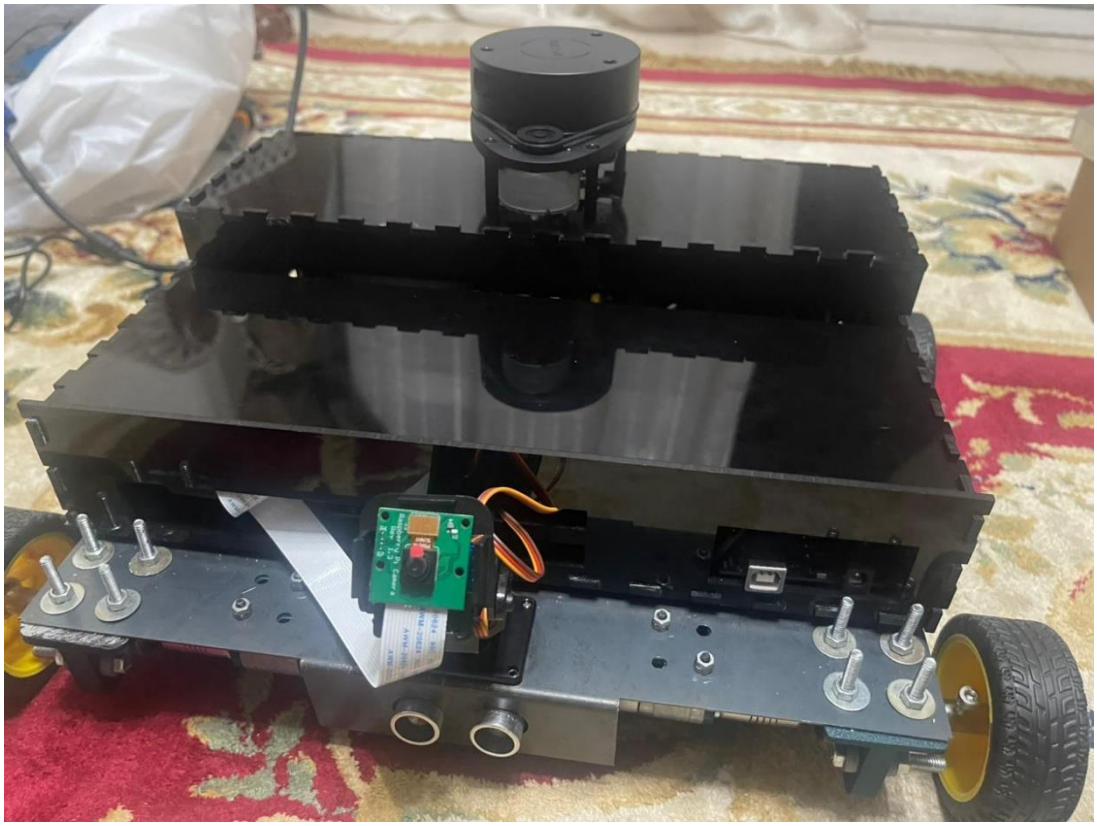
This code provides a basic framework for implementing PID-controlled motor speed using encoders. You may need to adjust the PID parameters and other settings based on your specific motor and system requirements.

12.0. Robot Hardware Implementation









13.0. Navigation

Navigation Using LiDAR and Hector SLAM

1. LiDAR (Light Detection and Ranging):

LiDAR is a remote sensing method that uses light in the form of a pulsed laser to measure variable distances to the Earth. These light pulses, combined with other data recorded by the system, generate precise, three-dimensional information about the shape of the Earth and its surface characteristics.

- **Components of LiDAR:**
  - **Laser:** Emits pulses of light.
  - **Scanner:** Directs the laser pulses and collects the reflected signals.
  - **GPS:** Provides accurate positioning data.
  - **IMU (Inertial Measurement Unit):** Measures the orientation of the sensor.
- **Applications:**
  - Autonomous vehicles for obstacle detection and navigation.
  - Topographic mapping and 3D modeling.
  - Environmental monitoring.

2. Hector SLAM (Simultaneous Localization and Mapping):

Hector SLAM is a method used for creating maps of an environment while simultaneously keeping track of the vehicle's location within that environment. It is particularly well-suited for real-time robotics applications because it does not rely on odometry or inertial measurements but uses the high update rate of LiDAR scans.

- **Key Features:**
  - **Robustness:** Performs well in environments with fast motion and without relying on wheel odometry.
  - **Real-Time Capability:** Suitable for high-speed operations due to its fast processing time.
  - **Accuracy:** Provides high-resolution maps and precise localization.

How They Work Together:

1. **Data Collection:** The LiDAR sensor continuously emits laser pulses and collects the reflected signals from the environment. This data forms a series of point clouds representing the surroundings.
2. **Scan Matching:** Hector SLAM uses scan matching techniques to align consecutive LiDAR scans. It identifies the best match between the current scan and the previously stored map.

3. **Map Updating:** As the vehicle moves, Hector SLAM updates the map in real-time by integrating new LiDAR data. It corrects any positional drift by comparing new scans to the existing map.
4. **Localization:** The system maintains an accurate estimate of the vehicle's position by continuously aligning incoming scans with the map. This enables the vehicle to navigate effectively through the environment.

Applications:

- **Autonomous Vehicles:** LiDAR and Hector SLAM are widely used in autonomous cars and drones for navigation, obstacle avoidance, and environment mapping.
- **Robotics:** Mobile robots use this combination for indoor and outdoor navigation, ensuring precise movement and interaction with their environment.
- **Surveying and Mapping:** Used in surveying applications to create detailed maps of complex environments.

By using LiDAR for accurate distance measurements and Hector SLAM for real-time mapping and localization, autonomous systems can navigate complex environments efficiently and safely.

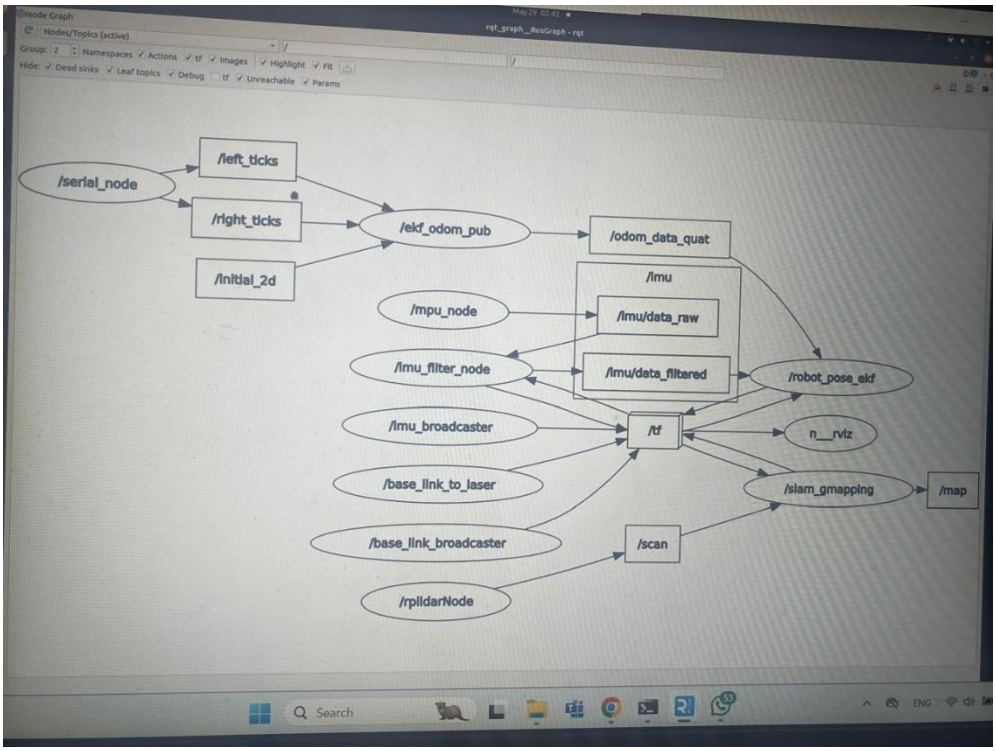


Figure 46 Nodes While mapping.

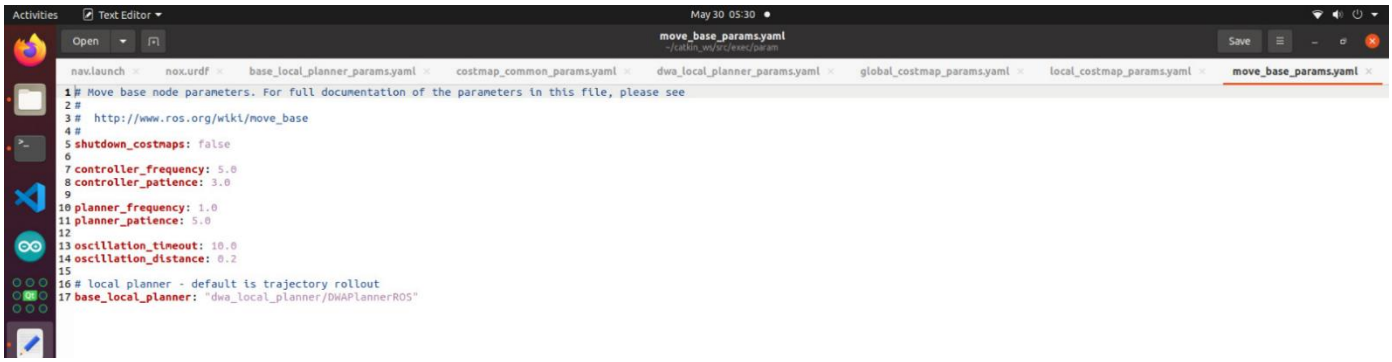


Figure 47 move base Yaml File

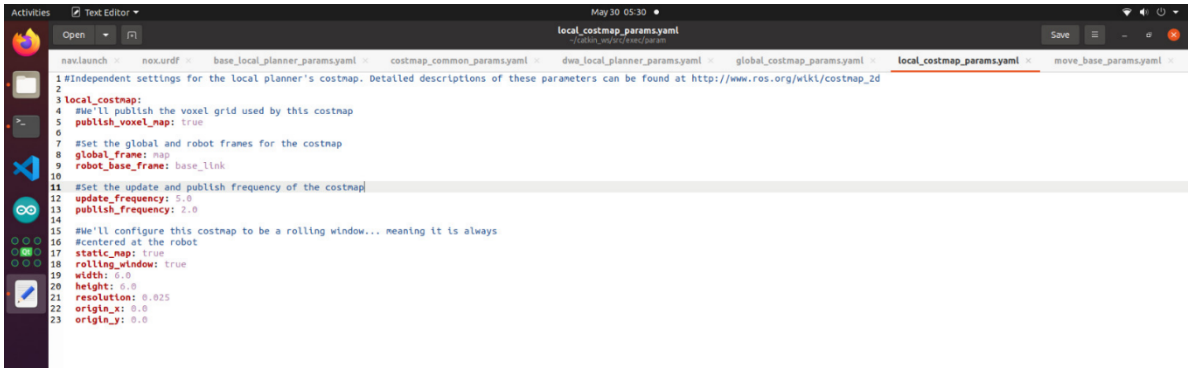


Figure 48 local Cost map Yaml File

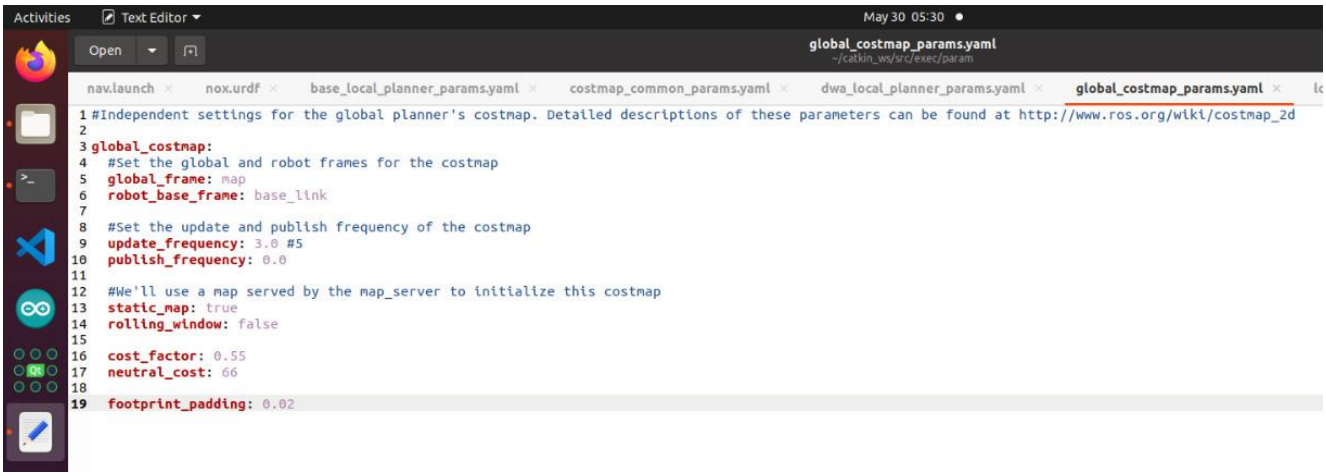


Figure 49 Global Cost map

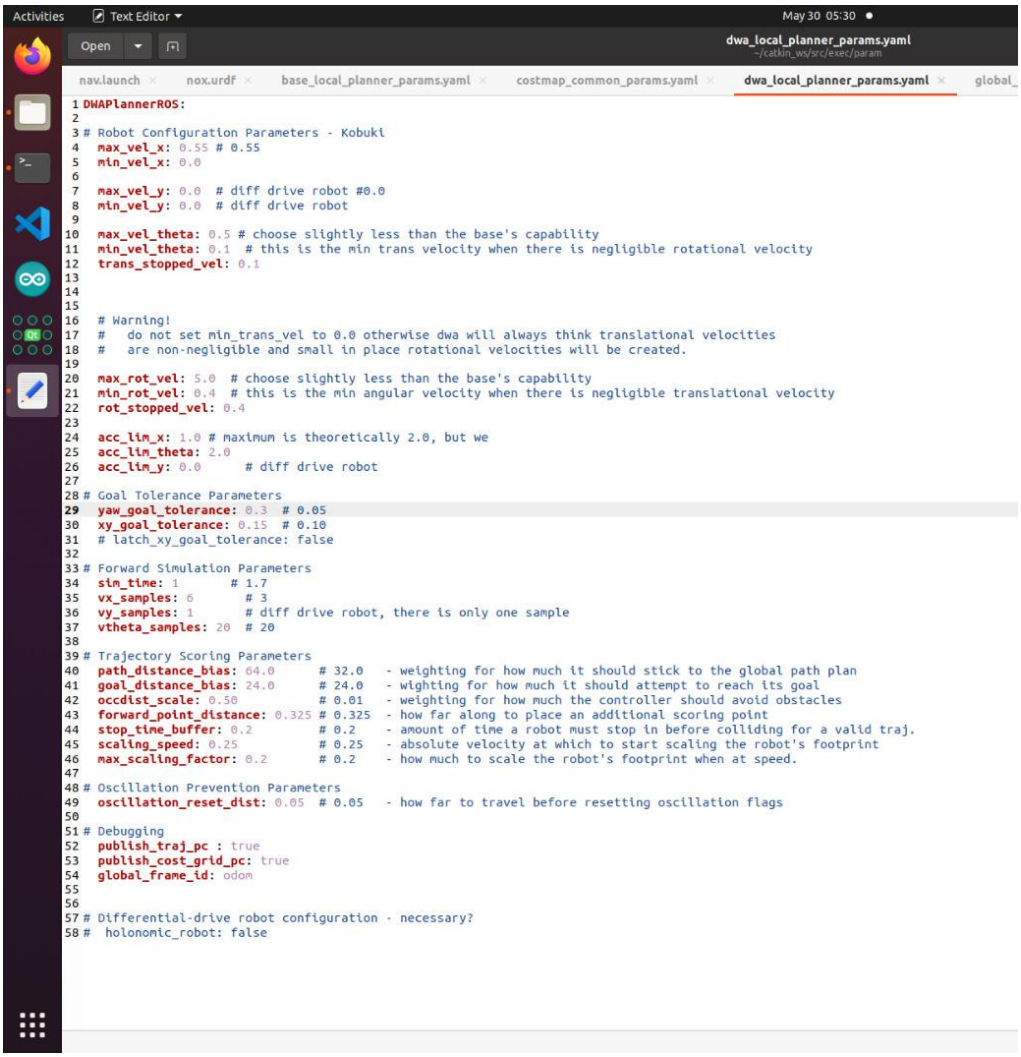


Figure 50 DWA local Planner.

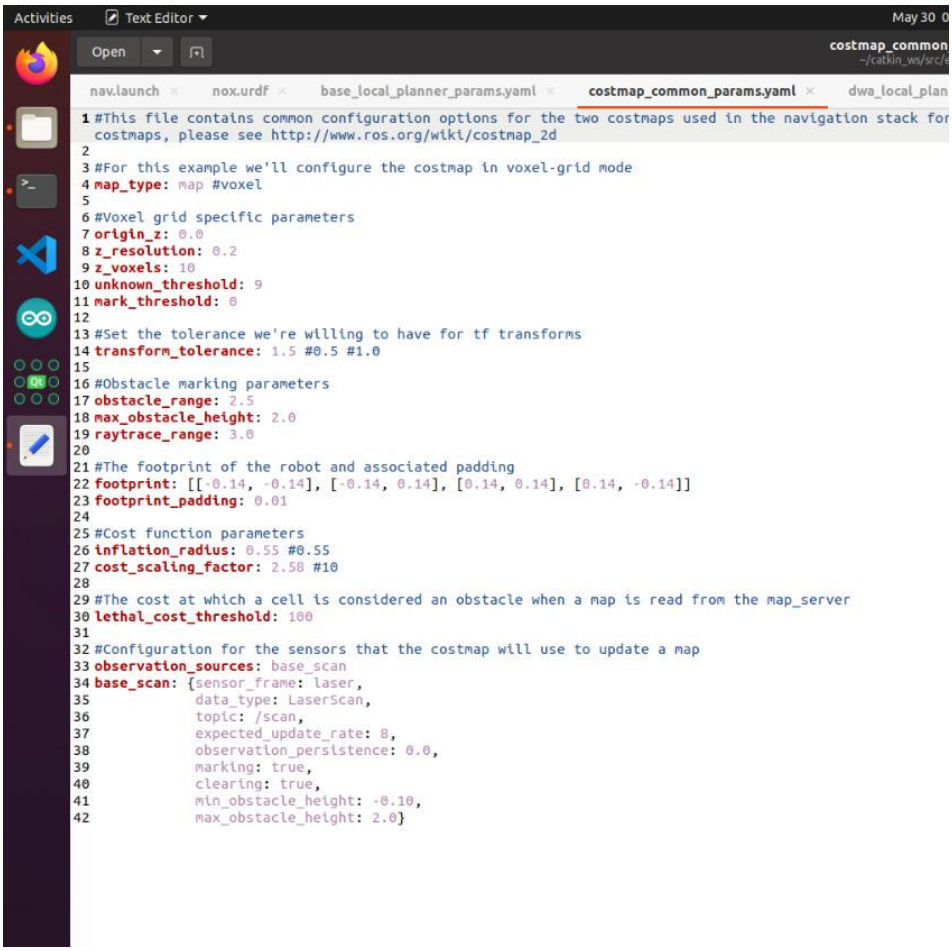


Figure 51 Cost map Common Parameters

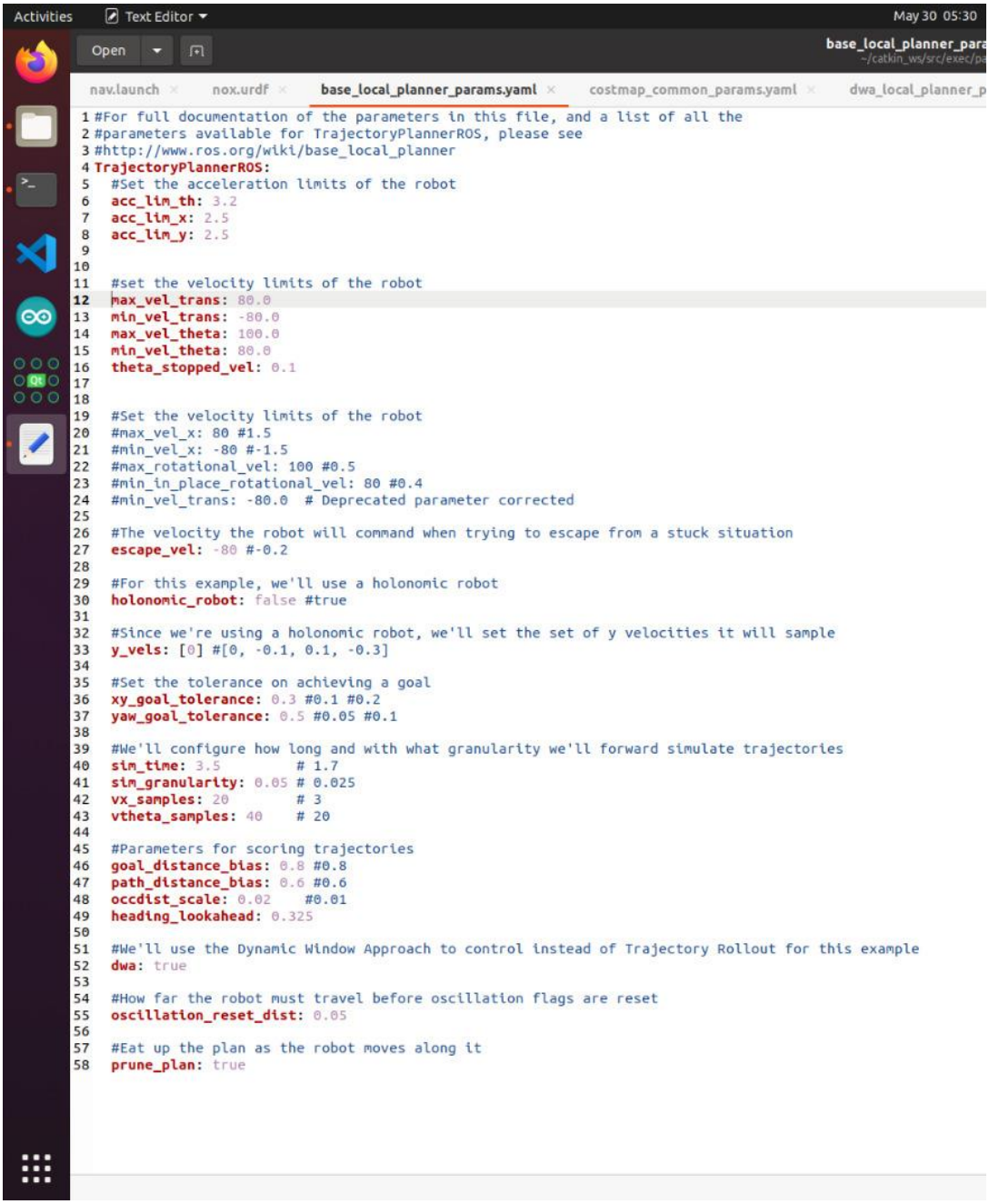


Figure 52 Base Local planner parameters

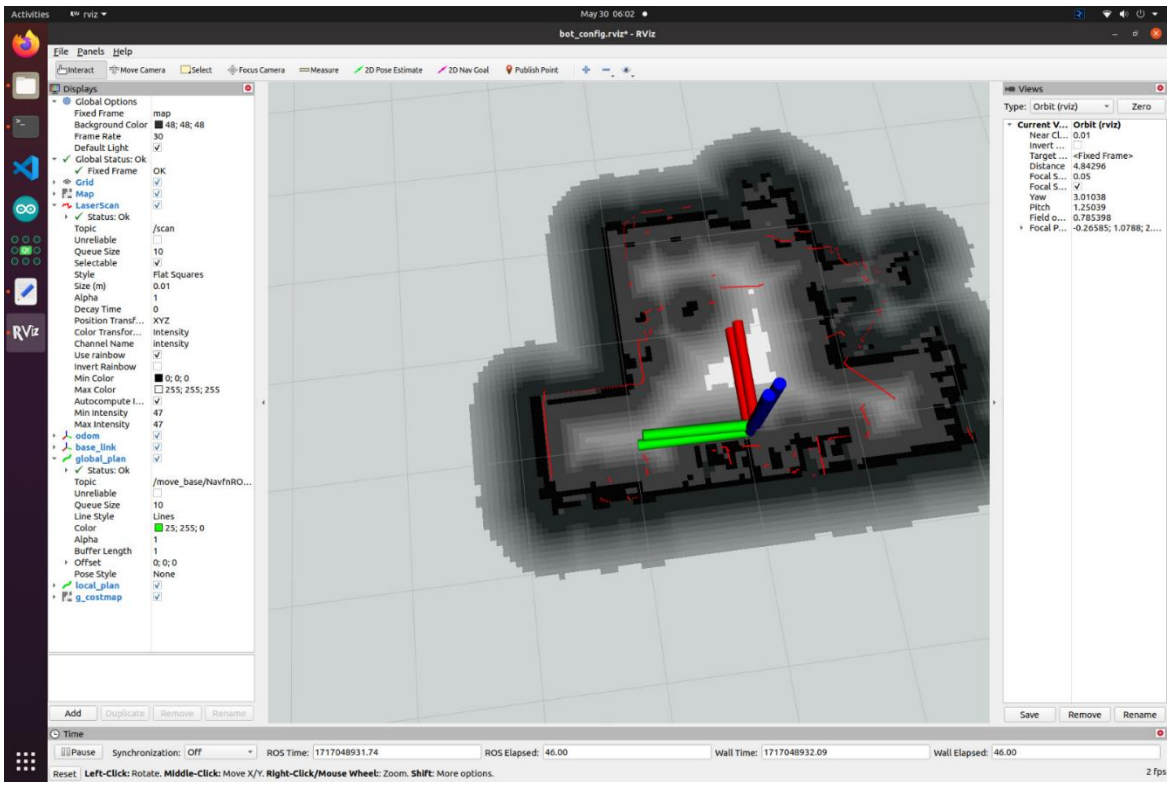


Figure 53 Room map.

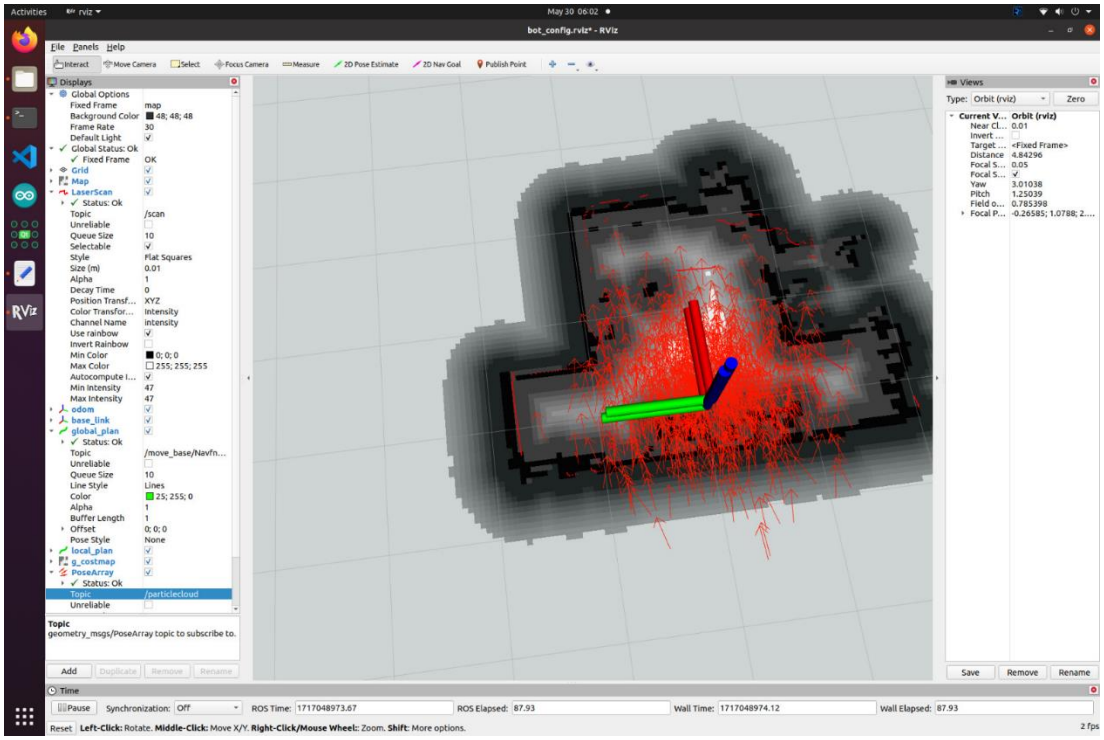


Figure 54 Point Cloud

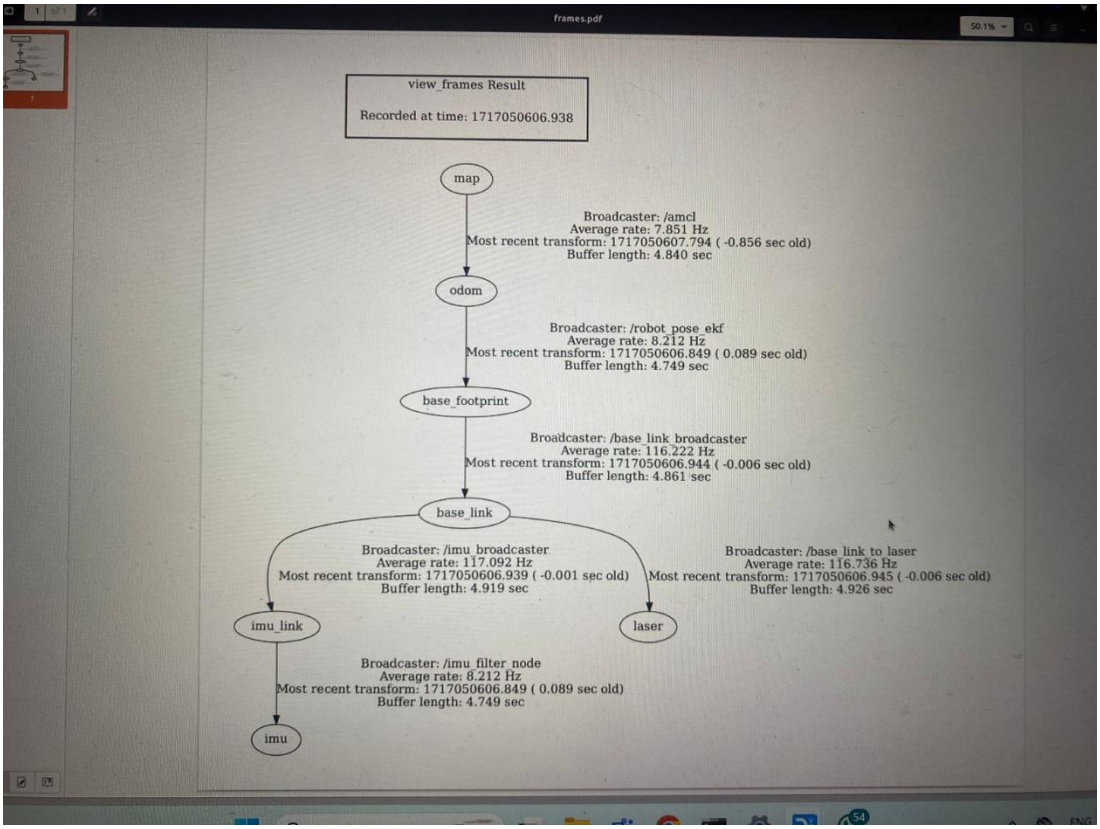


Figure 55 TF Frames relation diagram.

12.2. Code

Publishing ticks from both encoders to calculate odometery

```
#include <PID_v1_bc.h>


//#include <PID_v1.h>
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/Int16.h>

// Motor control pins
const int motor1PWM = 5;         // PWM pin for left motor
const int motor1Direction = 7;   // Direction pin for left motor
const int motor2PWM = 44;        // PWM pin for right motor
const int motor2Direction = 45;  // Direction pin for right motor

// Encoder pins
const int encoderPinA1 = 2;      // Encoder channel A for motor 1 connected to digital pin 2
const int encoderPinB1 = 3;      // Encoder channel B for motor 1 connected to digital pin 3
const int encoderPinA2 = 20;     // Encoder channel A for motor 2 connected to digital pin 20
const int encoderPinB2 = 21;     // Encoder channel B for motor 2 connected to digital pin 21

////////////////// Tick Data Publishing Variables and Constants ///////////////

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 20

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 3
```

```
#define ENC_IN_RIGHT_B 21

// PID Parameters
const double max_speed = 0.4;    // Max speed in m/s
const double speed_to_pwm_ratio = 0.00235;
const double min_speed_cmd = 0.0882;
const double PID_left_param[] = {0, 0, 0.1};
const double PID_right_param[] = {0, 0, 0.1};
double speed_req_left = 0;
double speed_act_left = 0;
double speed_cmd_left = 0;
double speed_req_right = 0;
double speed_act_right = 0;
double speed_cmd_right = 0;

// Variables for encoder position
volatile float pos_left = 0;       // Left motor encoder position
volatile float pos_right = 0;      // Right motor encoder position

PID PID_leftMotor(&speed_act_left, &speed_cmd_left, &speed_req_left, PID_left_param[0],
PID_left_param[1], PID_left_param[2], DIRECT);
PID PID_rightMotor(&speed_act_right, &speed_cmd_right, &speed_req_right, PID_right_param[0],
PID_right_param[1], PID_right_param[2], DIRECT);

ros::NodeHandle nh;

// Function that will be called when receiving command from host
void handle_cmd(const geometry_msgs::Twist& cmd_vel) {
  double angular_speed_req = cmd_vel.linear.x; // Switching linear and angular velocities
  double speed_req = cmd_vel.angular.z;       // Switching linear and angular velocities
  double wheelbase = 0.34;  // Wheelbase, in m
```

```
  speed_req_left = speed_req - angular_speed_req * (wheelbase / 2);
  speed_req_right = speed_req + angular_speed_req * (wheelbase / 2);
}


//////////////////////////////TICKS CALCULATIONS///////////////////////////////////

// Number of ticks a wheel makes moving a linear distance of 1 meter
// This value was measured manually.
const double TICKS_PER_METER = 2200; // Originally 2880

// Time interval for measurements in milliseconds
const int interval = 30;
long previousMillis = 0;
long currentMillis = 0;

// Record the time that the last velocity command was received
double lastCmdVelReceived = 0;

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;


// Publishers for encoder ticks
// Minumum and maximum values for 16-bit integers
// Range of 65,535
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;
```

```
// Set linear velocity and PWM variable values for each wheel
double velLeftWheel = 0;
double velRightWheel = 0;
double pwmLeftReq = 0;
double pwmRightReq = 0;

//---------TICKS PUBLISHERS---------/
// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks", &left_wheel_tick_count);


////////////////////// Tick Data Publishing Functions ///////////////////////

// Increment the number of ticks
void right_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_B);

  if (val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }
```

```
  if (Direction_right) {

    if (right_wheel_tick_count.data == encoder_maximum) {
      right_wheel_tick_count.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count.data++;
    }
  }
  else {
    if (right_wheel_tick_count.data == encoder_minimum) {
      right_wheel_tick_count.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count.data--;
    }
  }
}

// Increment the number of ticks
void left_wheel_tick() {

  // Read the value for the encoder for the left wheel
  int val = digitalRead(ENC_IN_LEFT_B);

  if (val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
```

```
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count.data == encoder_maximum) {
      left_wheel_tick_count.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count.data++;
    }
  }
  else {
    if (left_wheel_tick_count.data == encoder_minimum) {
      left_wheel_tick_count.data = encoder_maximum;
    }
    else {
      left_wheel_tick_count.data--;
    }
  }
}

////////////////////// Motor Controller Functions ///////////////////////////

// Calculate the left wheel linear velocity in m/s every time a
// tick count message is rpublished on the /left_ticks topic.
void calc_vel_left_wheel(){

  // Previous timestamp
  static double prevTime = 0;
```

```
  // Variable gets created and initialized the first time a function is called.
  static int prevLeftCount = 0;

  // Manage rollover and rollunder when we get outside the 16-bit integer range
  int numOfTicks = (65535 + left_wheel_tick_count.data - prevLeftCount) % 65535;

  // If we have had a big jump, it means the tick count has rolled over.
  if (numOfTicks > 10000) {
       numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velLeftWheel = numOfTicks/TICKS_PER_METER/((millis()/1000)-prevTime);

  // Keep track of the previous tick count
  prevLeftCount = left_wheel_tick_count.data;

  // Update the timestamp
  prevTime = (millis()/1000);

}

// Calculate the right wheel linear velocity in m/s every time a
// tick count message is published on the /right_ticks topic.
void calc_vel_right_wheel(){

  // Previous timestamp
  static double prevTime = 0;

  // Variable gets created and initialized the first time a function is called.
```

```
  static int prevRightCount = 0;

  // Manage rollover and rollunder when we get outside the 16-bit integer range
  int numOfTicks = (65535 + right_wheel_tick_count.data - prevRightCount) % 65535;

  if (numOfTicks > 10000) {
       numOfTicks = 0 - (65535 - numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  velRightWheel = numOfTicks/TICKS_PER_METER/((millis()/1000)-prevTime);

  prevRightCount = right_wheel_tick_count.data;

  prevTime = (millis()/1000);

}



////////////////////////////////////////////////////////////////////////////////

ros::Subscriber<geometry_msgs::Twist> cmd_vel("cmd_vel", handle_cmd);

void setup() {
  pinMode(motor1PWM, OUTPUT);      // Set pins for left motor as output
  pinMode(motor1Direction, OUTPUT);
  pinMode(motor2PWM, OUTPUT);      // Set pins for right motor as output
  pinMode(motor2Direction, OUTPUT);
```

```
  // Set pin states of the encoder
  pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
  pinMode(ENC_IN_LEFT_B , INPUT);
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);

  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A), left_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A), right_wheel_tick, RISING);

  nh.initNode();
  nh.getHardware()->setBaud(57600);
  nh.subscribe(cmd_vel);
  //Initialize Publishers
  nh.advertise(rightPub);
  nh.advertise(leftPub);



  // Setting PID parameters
  PID_leftMotor.SetSampleTime(95);
  PID_rightMotor.SetSampleTime(95);
  PID_leftMotor.SetOutputLimits(-max_speed, max_speed);
  PID_rightMotor.SetOutputLimits(-max_speed, max_speed);
  PID_leftMotor.SetMode(AUTOMATIC);
  PID_rightMotor.SetMode(AUTOMATIC);
}

void loop() {
  nh.spinOnce();
```

```
  PID_leftMotor.Compute();
  PID_rightMotor.Compute();

  int PWM_leftMotor = constrain(((speed_req_left + sgn(speed_req_left) * min_speed_cmd) / speed_to_pwm_ratio) + (speed_cmd_left / speed_to_pwm_ratio), -255, 255);
  int PWM_rightMotor = constrain(((speed_req_right + sgn(speed_req_right) * min_speed_cmd) / speed_to_pwm_ratio) + (speed_cmd_right / speed_to_pwm_ratio), -255, 255);

  if (speed_req_left == 0) {
    analogWrite(motor1PWM, 0);
  } else if (PWM_leftMotor > 0) {
    digitalWrite(motor1Direction, HIGH);
    analogWrite(motor1PWM,abs(PWM_leftMotor));
  } else {
    digitalWrite(motor1Direction, LOW);
    analogWrite(motor1PWM, abs(PWM_leftMotor));
  }

  if (speed_req_right == 0) {
    analogWrite(motor2PWM, 0);
  } else if (PWM_rightMotor > 0) {
    digitalWrite(motor2Direction, HIGH);
    analogWrite(motor2PWM, abs(PWM_rightMotor));
  } else {
    digitalWrite(motor2Direction, LOW);
    analogWrite(motor2PWM, abs(PWM_rightMotor));
  }

  // Record the time
  currentMillis = millis();
```

```
  // If the time interval has passed, publish the number of ticks,
  // and calculate the velocities.
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

    // Publish tick counts to topics
    leftPub.publish( &left_wheel_tick_count );
    rightPub.publish( &right_wheel_tick_count );

    // Calculate the velocity of the right and left wheels
    calc_vel_right_wheel();
    calc_vel_left_wheel();

  }

  delay(100); // Loop delay to control execution rate
}

/*
void encoderLeftMotor() {
  if (digitalRead(encoderPinA1) == digitalRead(encoderPinB1)) pos_left++;
  else pos_left--;
}

void encoderRightMotor() {
  if (digitalRead(encoderPinA2) == digitalRead(encoderPinB2)) pos_right--;
  else pos_right++;
}*/
```

```
int sgn(double x) {
  return(x>0)-(x<0);
}
```

14.0. GUI

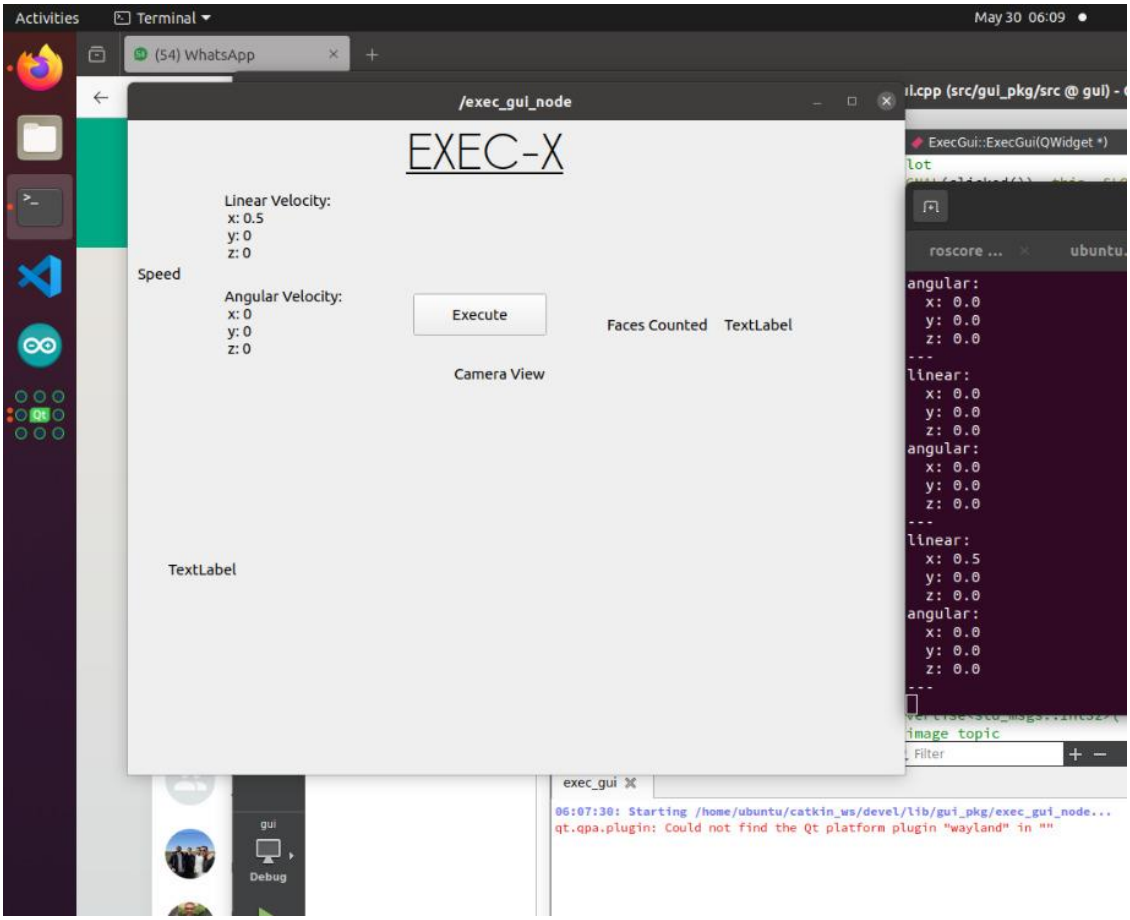


Figure 56 GUI interface.

1) Execute Button is used launch a laser in case an enemy was detected

2) Linear and angular velocities are the commands coming from navigation stack to move the robot.

3) Camera view is the display live stream of camera

13.2. Code

```
#include <ros.h>
#include <std_msgs/Int32.h>

#define Laser  13

ros::NodeHandle nh;

bool execute_status_high = false;
bool face_detector_status_high = false;

void executeMessageCb(const std_msgs::Int32& status_msg) {
  if (status_msg.data == 1) {
    execute_status_high = true;
  } else {
    execute_status_high = false;
  }
  updateLaserState();
}

void faceDetectorMessageCb(const std_msgs::Int32& status_msg) {
  if (status_msg.data == 1) {
    face_detector_status_high = true;
  } else {
    face_detector_status_high = false;
  }
  updateLaserState();
}
```

```
void updateLaserState() {
  if (execute_status_high && face_detector_status_high) {
    digitalWrite(Laser, HIGH); // Turn on laser
  } else {
    digitalWrite(Laser, LOW);  // Turn off laser
  }
}

ros::Subscriber<std_msgs::Int32> execute_sub("execute/status", &executeMessageCb);
ros::Subscriber<std_msgs::Int32> face_detector_sub("face_detector/status", &faceDetectorMessageCb);

void setup() {
  pinMode(Laser, OUTPUT);
  nh.initNode();
  nh.subscribe(execute_sub);
  nh.subscribe(face_detector_sub);
}

void loop() {
  nh.spinOnce();
  delay(1);
}
```

While Face was detected and the execute button was pressed a laser module is turned to kill the enemy.

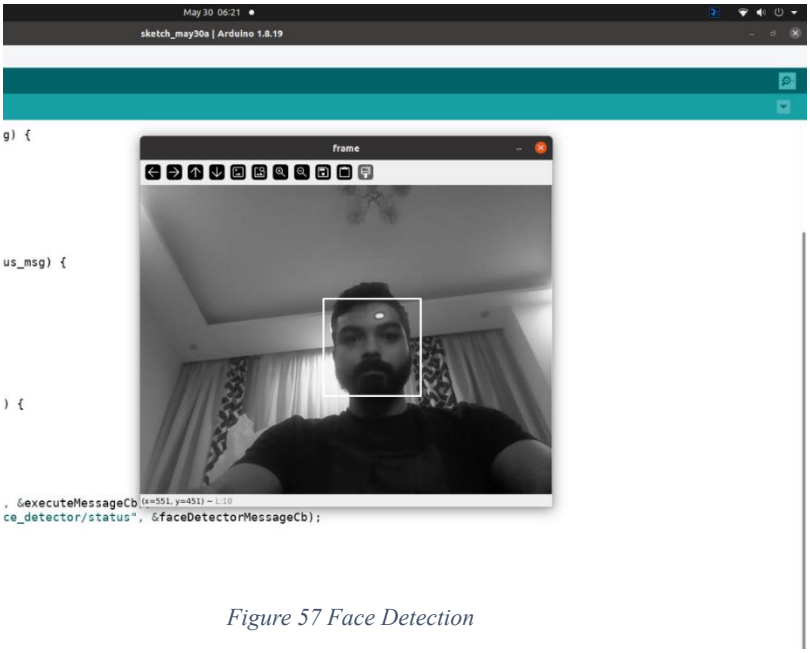


Figure 57 Face Detection

13.0. REFERENCES

Simulating Autonomously Navigating Robot using ROS:

https://www.youtube.com/watch?v=DU31PuQjK-I

Building a mobile robot – Articulated Robotics:

https://www.youtube.com/playlist?list=PLunhqkrRNRhYAffV8JDiFOatQXuU-NnxT

How to Set Up the ROS Navigation Stack on a Robot - Automatic Addison

https://automaticaddison.com/how-to-set-up-the-ros-navigation-stack-on-a-robot/

A Drive you can find in it more of:

CAD SOURCE FILES

SIMULATION FILES

PDF (reference)

Drive Link: https://drive.google.com/drive/folders/1Cuw5LCvbwBuZLoSFR-H01rKJmDRA2tTd