Faculty of Engineering Ain Shams University
International credit hours education program (ICHEP)
Mechatronics senior 1

Design of mechatronics systems 2

MCT332

Name	ID
Habiba Tamer	20p4355
Omar Mohamed	20P4257
Zeyad Mohamed	20P4913
Mohamed Mahmoud	20P4202
Bishoy Girgis	20P5707
Mahmoud Fouad	20P3839

Submitted to: Dr. Shady Ahmed

Abstract

This project outlines the design and implementation of an autonomous lawn mower robot equipped with obstacle detection and avoidance capabilities, a QR code-controlled cutting blade, and a user interface application for monitoring and control. The lawn mower robot utilizes ultrasonic sensors to detect obstacles in its path, ensuring safe navigation in garden environments. Upon encountering an obstacle, the robot adjusts its course to prevent collisions.

A cutting blade is integrated into the robot, with its operation controlled via a Raspberry Pi camera that scans QR codes. This mechanism allows the cutting blade to be activated or deactivated by scanning the appropriate QR code, providing a user-friendly and secure method for managing the blade's operation.

Additionally, a graphical user interface (GUI) application has been developed for users to interact with the robot. Users are required to log in with a username and password, ensuring secure access to the application. The application communicates with the Raspberry Pi to receive real-time status updates of the cutting blade, displaying whether the blade is currently in "Mowing On" or "Mowing Off" mode.

The integration of obstacle detection, QR code-controlled blade operation, and a user-friendly GUI provides an efficient and safe solution for automated lawn maintenance, enhancing user experience and ensuring reliable performance.

Table of Contents

List of figures

1.0. INTRODUCTION

In response to the growing need for adaptable and intelligent robotic systems, this project focuses on the creation of a semi-autonomous mobile robot. Combining autonomous navigation with the ability to incorporate human input, our goal is to develop a versatile robotic platform capable of efficient operation in dynamic environments. This report outlines the design, development, and implementation of the robot, emphasizing key components, sensors, and algorithms. By merging autonomy and user control, our project aims to deliver a functional prototype with potential applications in diverse settings, from industrial facilities to everyday households. This work not only contributes to the academic understanding of semi-autonomous robotics but also holds promise for practical deployment in real-world scenarios.

2.0. OBJECTIVE

It is required to design a mobile robot that would achieve a certain task or goal, fulfill a certain need or can be of assistance in completing a specified objective.

The constraints for this mobile robot's design are as follows:

- **For the mechanical design**
  - The robot should be mainly constructed of metal.
  - Mechanical design methodology should be applied when designing shafts, bearing placements, wheel configurations, lifting mechanisms (if any)
- **Navigation**
  - Semi-Autonomous: The robot should be able to observe, process information and take a decision e.g., avoid an obstacle or pickup an object without any human interaction or aid, no remotes or joysticks can be used to navigate the robot.
- **Functionality**
  - All robots must contain a computer vision module that either aids in objective completion or is incorporated into the robot's navigation system.
  - PID control must be implemented in your motion actuators' low-level control.
- **Other**
  - The robot must include at least one micro-controller board that is designed by the team.
  - You must provide a GUI application on a PC using **QT** that communicates with the robot using WiFi.

3.0. ROBOT FUNCTION

We decided to make ***a Mower robot.***

A mower robot is an autonomous robotic device equipped with a cutting blade designed for the purpose of automating lawn maintenance. Utilizing sensors for navigation and obstacle avoidance, it can efficiently mow grass in a predefined area. Mower robots provide a convenient and time-saving solution for lawn care, requiring minimal human intervention once set up and programmed.

Figure 1 , Mower robot.

4.0. ROBOT DESIGN

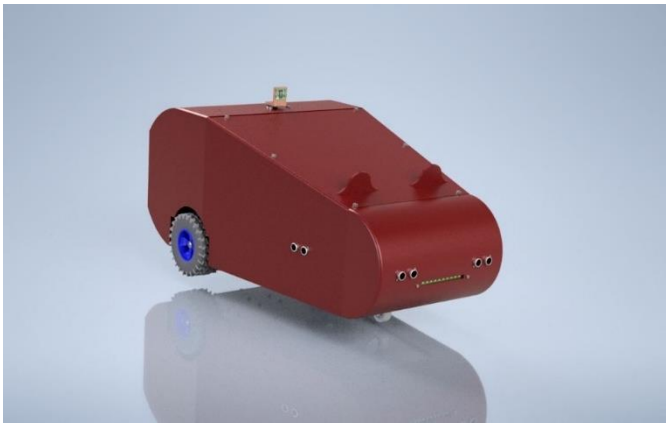


Figure 2 , First CAD render

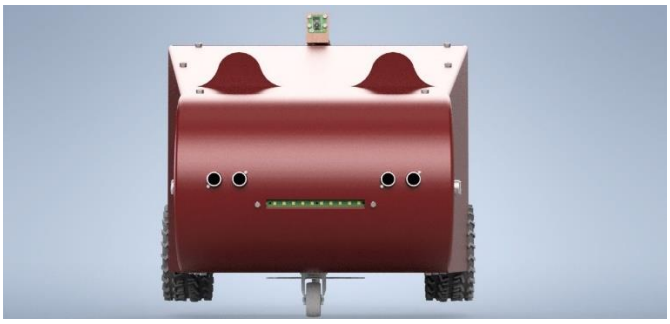


Figure 3 , second CAD render

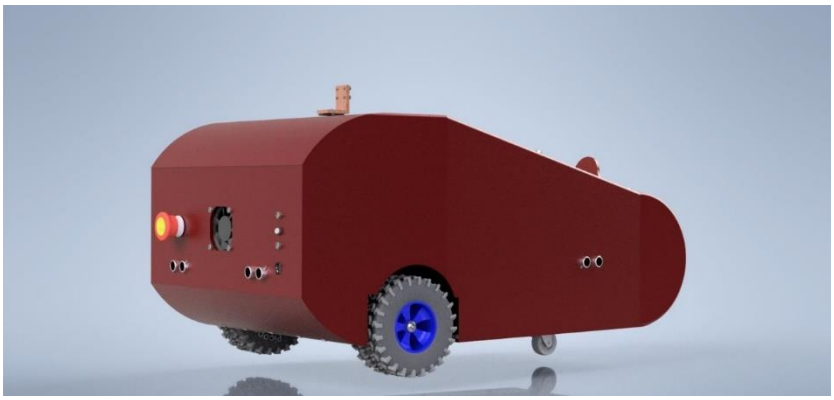


Figure 4 , Third CAD render

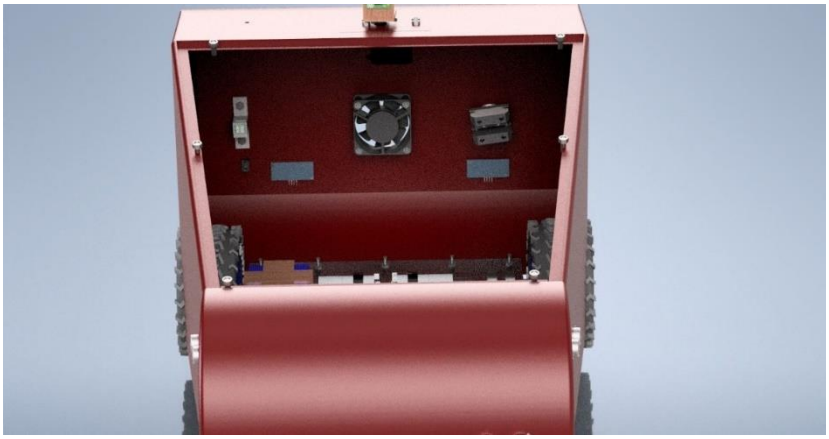


Figure 5 , Fourth CAD render

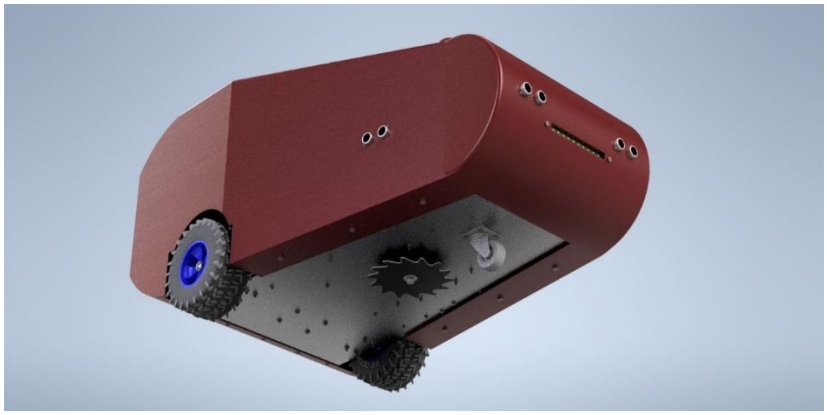


Figure 6 , Fifth CAD render

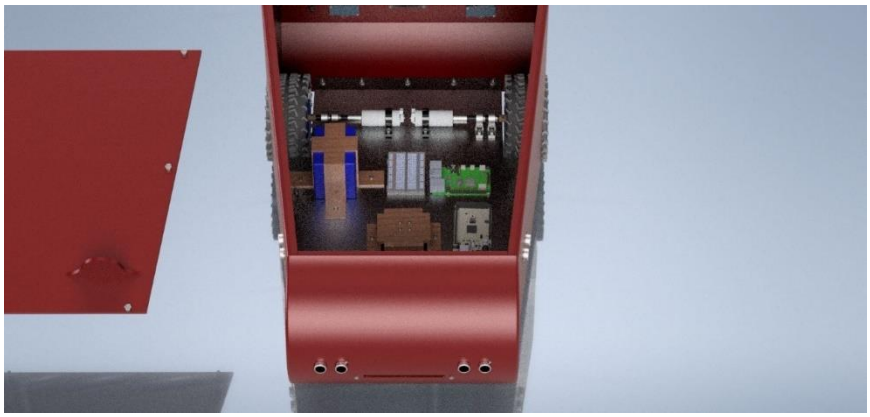


Figure 7 , Sixth CAD render

Figure 8 , Seventh CAD render

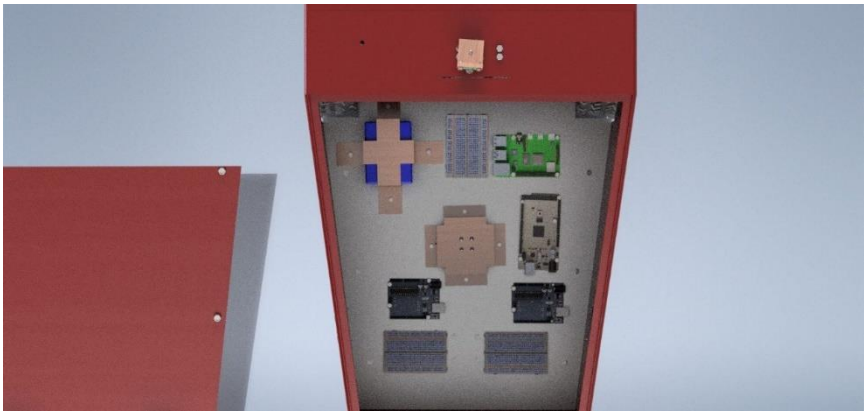


Figure 9 , Eighth CAD render

5.0. COMPONENTS

5.1. Robot Wheels

The robot wheels serve as the primary means of mobility, allowing the robot to move efficiently and navigate its environment.

Figure 10 , Robot Wheels

5.2. Arduino Mega

The Arduino Mega acts as the microcontroller, governing various robotic functions such as motor control, sensor interfacing, and decision-making algorithms.

Figure 11 , Arduino mega

5.3. Raspberry pi 4

The Raspberry Pi 4 serves as the brain of the robot, handling higher-level tasks such as image processing, communication, and overall system coordination.

Figure 12 , raspberry pi 4

5.4. Lipo Battery

The LiPo battery serves as the power source for the robot, supplying the necessary energy for motor operation, electronic components, and overall system functionality.

Figure 13 , lipo battery

5.5. DC Brushless Motor

Brushless DC motors do not use brushes. With brushed motors, the brushes deliver current through the commutator into the coils on the rotor. So how does a brushless motor pass current to the rotor coils? It doesn't—because the coils are not located on the rotor.

Figure 14 , DC Brushless motor

5.6. Servo Motor

The servo motor is employed for precise and controlled rotational movement, commonly used in robotic applications for tasks such as manipulating arms or implementing specific angles.

Figure 15 , servo motor

5.7. Raspberry pi camera

The Raspberry Pi Camera captures visual data, enabling the robot to perceive and interpret its surroundings for tasks such as object detection, navigation, and decision-making.

Figure 16 , raspberry pi camera

5.8. Ultrasonic sensor

The ultrasonic sensor enables the robot to detect obstacles and measure distances by emitting and receiving ultrasonic waves, contributing to obstacle avoidance and navigation.

Figure 17 , ultrasonic sensor

5.9. Cooling Electric Fan

The cooling electric fan is essential for regulating the temperature within the robot's components, preventing overheating and ensuring optimal performance during extended operation.

Figure 18, cooling electric fan

5.10. Emergency Button

The emergency button serves as a safety feature, providing a quick and reliable way to halt all robot operations in case of unforeseen circumstances or emergencies.

Figure 19 , emergency button

5.11. DC barrel jack adapter

The DC barrel jack provides a standardized power input, allowing easy connection of the power source (such as the LiPo battery) to the robot's electrical system for consistent and reliable power supply.

Figure 20 , DC barrel jack

5.12. Self-Aliening flange bearing

The self-aligning bearing enhances the robot's mechanical structure by allowing flexible movement and accommodating misalignments, contributing to smoother and more efficient overall operation.

Figure 21 , Self-aliening bearing

5.13. Hex Coupler

The hex coupler connects and synchronizes components, such as a motor shaft to a wheel or other mechanical parts, ensuring a reliable and efficient transfer of rotational motion.

Figure 22 , hex coupler

5.14. Caster wheel

The caster wheel provides stability and facilitates smooth turning for the robot, enhancing its maneuverability and overall control.

Figure 23 , caster wheel

5.15. Cutter blade

The grass blade cutter is a specialized attachment that equips the robot for tasks such as lawn maintenance. It enables the robot to autonomously trim or cut grass, expanding its functionality beyond basic mobility and sensing capabilities.

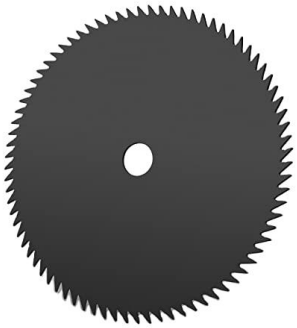


Figure 24 , cutter blade

6.0. STRESS ANALYSIS

We make a stress analysis using inventor on the main Aluminum chassis of the robot which carry the Electronic Components as arduinos , raspberry pi , Bearings , Breadboards & Motors and here is the chassis with all the components settled on it

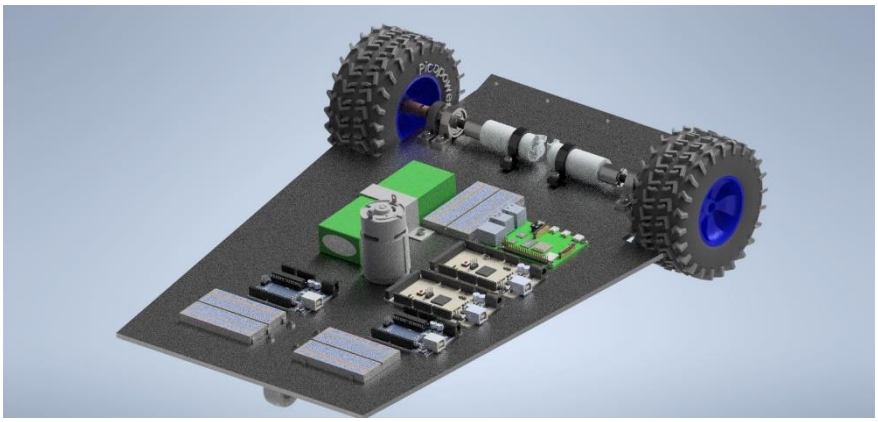


Figure 25: Chassis with Components

And what we have done on the stress analysis that we apply a fixation on the chassis from the caster wheel point and the driving rear wheels bearing and then we apply a force of 5 N which is approximately equals to the weight of all components and motors on the chassis and we start the simulation as shown below

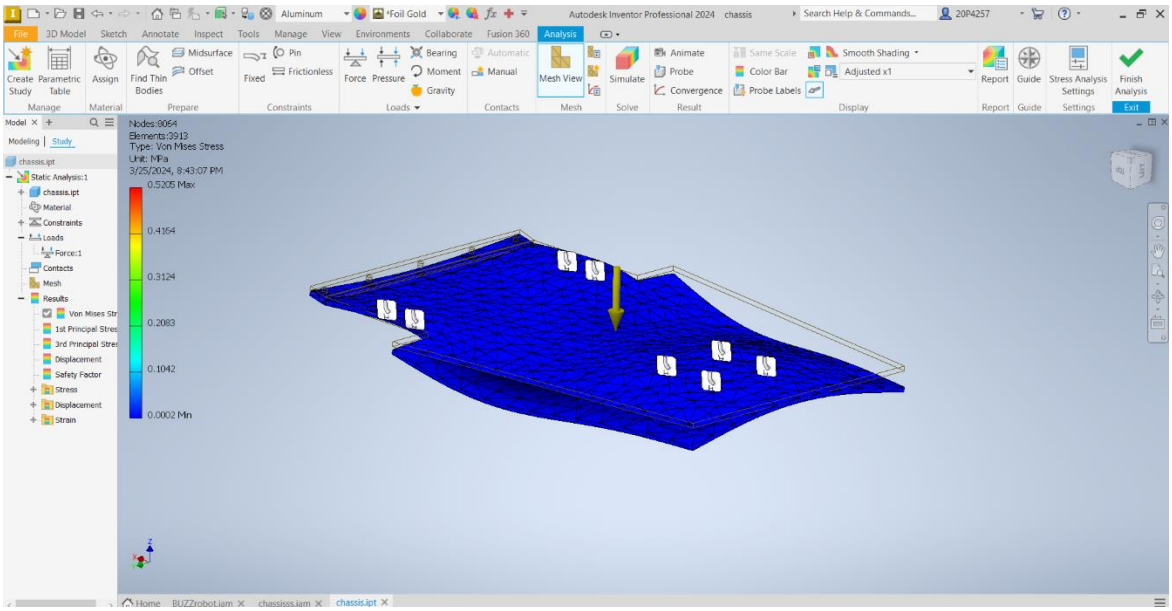


Figure 26: Stress Analysis 1

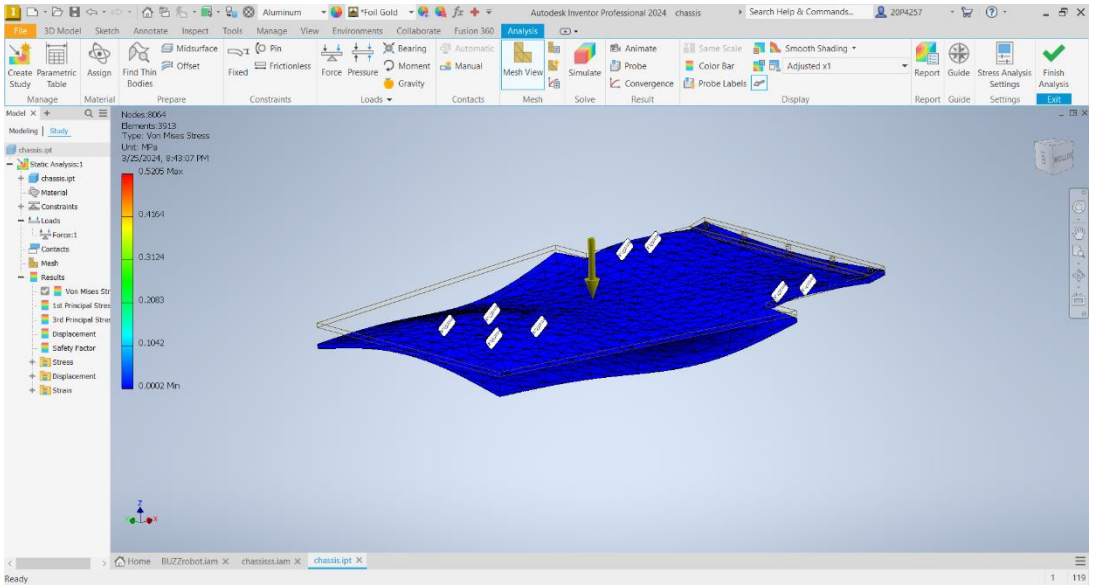


Figure 27: Stress Analysis 2

And here is the stress analysis report from inventor

Results

Reaction Force and Moment on Constraints

Constraint Name	Reaction Force		Reaction Moment	
	Magnitude	Component (X,Y,Z)	Magnitude	Component (X,Y,Z)
Fixed Constraint:1	5 N	0 N	0.0746982 N m	0 N m
		0 N		0.0746982 N m
		5 N		0 N m

Result Summary

Name	Minimum	Maximum
Volume	888041 mm^3	
Mass	2.39771 kg	
Von Mises Stress	0.000175332 MPa	0.520499 MPa
1st Principal Stress	-0.152471 MPa	0.801515 MPa
3rd Principal Stress	-0.48252 MPa	0.246943 MPa
Displacement	0 mm	0.00199927 mm
Safety Factor	15 ul	15 ul
Stress XX	-0.231975 MPa	0.431312 MPa
Stress XY	-0.0895619 MPa	0.0934501 MPa
Stress XZ	-0.210782 MPa	0.223358 MPa
Stress YY	-0.247615 MPa	0.448662 MPa
Stress YZ	-0.257751 MPa	0.261457 MPa
Stress ZZ	-0.36348 MPa	0.503283 MPa
X Displacement	-0.0000440889 mm	0.0000419778 mm
Y Displacement	-0.0000405771 mm	0.0000385611 mm
Z Displacement	-0.00199919 mm	0.00011212 mm
Equivalent Strain	0.00000000230201 ul	0.00000743106 ul
1st Principal Strain	0.00000000108414 ul	0.00000884673 ul
3rd Principal Strain	-0.00000535604 ul	-0.00000000119452 ul
Strain XX	-0.0000027714 ul	0.00000303727 ul
Strain XY	-0.00000172884 ul	0.0000018039 ul
Strain XZ	-0.00000406879 ul	0.00000431155 ul
Strain YY	-0.00000263648 ul	0.00000281483 ul
Strain YZ	-0.00000497545 ul	0.000005047 ul
Strain ZZ	-0.00000344268 ul	0.00000378531 ul

Figure 28: Stress Analysis Inventor Report

Also, there some additional Photos of the simulation as shown below

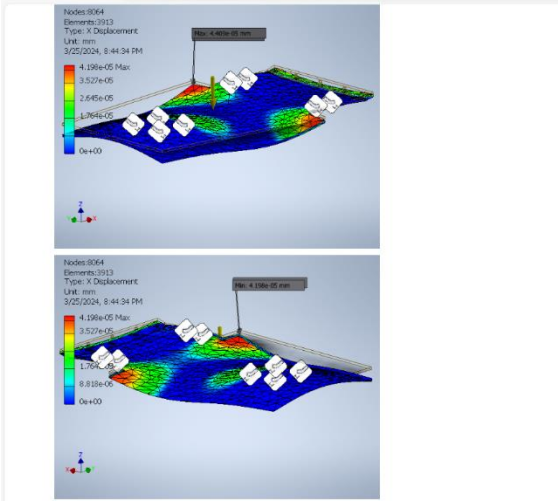


Figure 30: S1

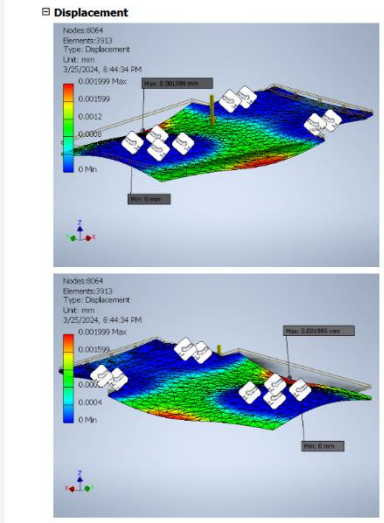


Figure 29:S2

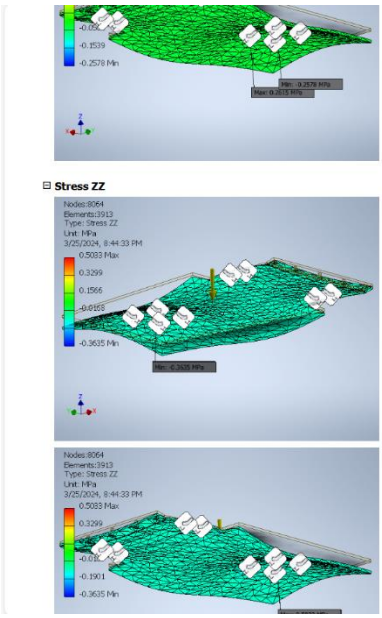


Figure 31: S3

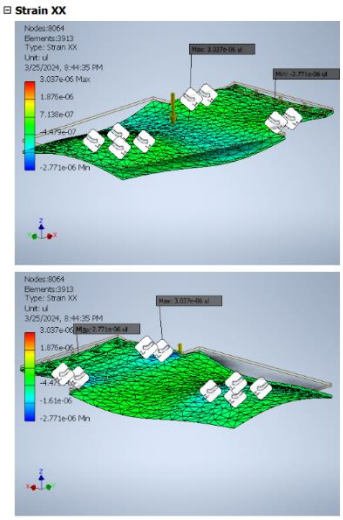


Figure 32: S4

7.0. ACTUATOR SIZING

7.1. Software

In actuator sizing we used solidworks to do it by making motor wheels 100 RPM and then we show the motion study diagram which was perfect as needed and we added an RMS sensor and it was shown as 160.1 N.mm which is 1.63 kg.cm, so we need to buy dc motors with torque higher than 1.63 kg.cm to make the robot move as desired.

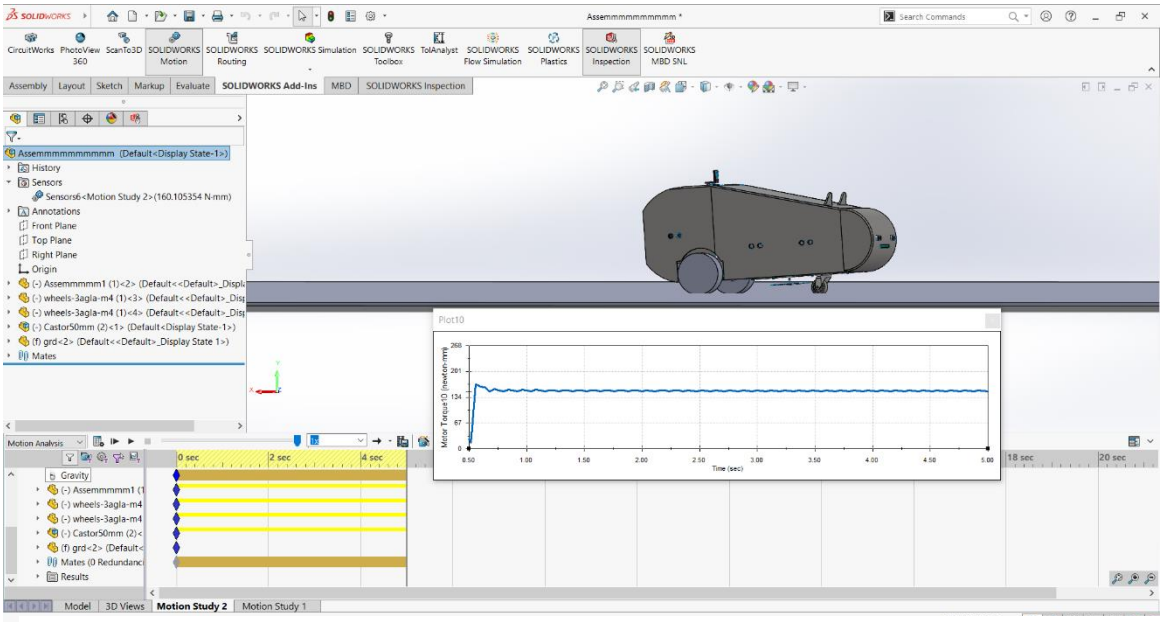


Figure 33: Motion Study with RMS Value Shown

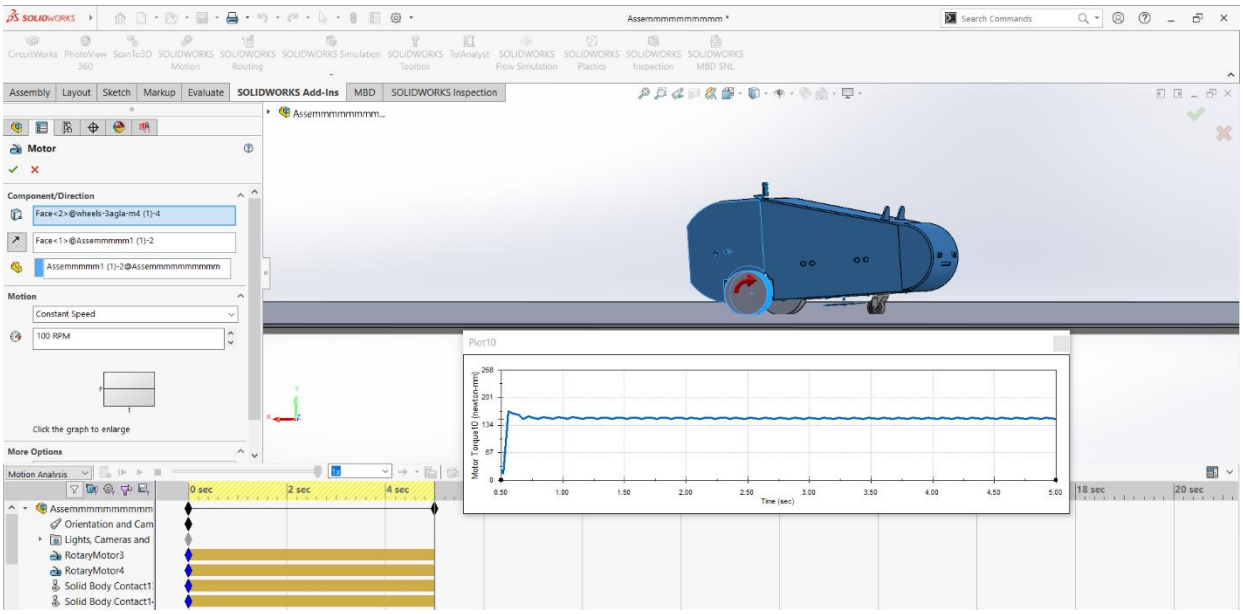


Figure 34: Motion Study with Motor Speed Value Shown

7.2. By hand

➢ Our Parameters:

- R_w=0.065
- μ_r=0.05
- μ_f=0.35
- C_d=0.1
- A_f=0.089
- V= 0.25 m/s
- T=0.3 s
- G=9.81
- $M_{total/wheel}$=2.7 kg
- η=0.9
- P=1.2

F_{air}= 1/2 * C_d * P * A_f * V^2

= ½ (0.1)(1.2)(0.089)$(0.3)^2$ = 3.3 x 10^{-4} N

F_R= μ_r *M g

= (0.05)(2.7)(9.81) = 1.324 N

J_{Motor}= $J_{LoadEff}$ = $\frac{\text{JeqWheel}+Jeqlinermass}{\eta}$

$J_{eqWheels}$=0.5M_{wheel} * R_w^2= 3.65x10^{-4}

$J_{eqlinearMass}$= M * R_w^2

=2.7$(0.065)^2$ = 0.0114

$J_{loadEff}=\frac{0.0114+3.65x10^{-4}}{0.9}$=0.013

J_{total} = $J_{eqWheels}$ + $J_{eqlinearMass}$ + $J_{loadEff}$

= 0.013 +0.0114 + 3.65x10^{-4} = 0.025

$\dot{\alpha} = \frac{\frac{V}{Rw}}{T} = \frac{\frac{0.25}{0.065}}{0.3}$ = 12.8

T_R = (F_{air} + F_r) R_w

=(3.3x10^{-4} +1.324)0.065 = 0.086 N.m

$T_{motor} = T_R + J_{total} * \dot{\alpha}$

At +acc = 0.086 +12.8(0.025) = 0.406 N.m

At 0 acc = 0.086+0(0.025) = 0.086 N.m

At -acc = 0.086 +12.8(-0.025) = -0.234

$T_{rms} = \sqrt{\frac{(0.406)^2*0.3+(0.086)^2*0.6+(-0.234)^2*0.3}{1.2}}$

=0.265 N.m = 2.65 Kg.cm

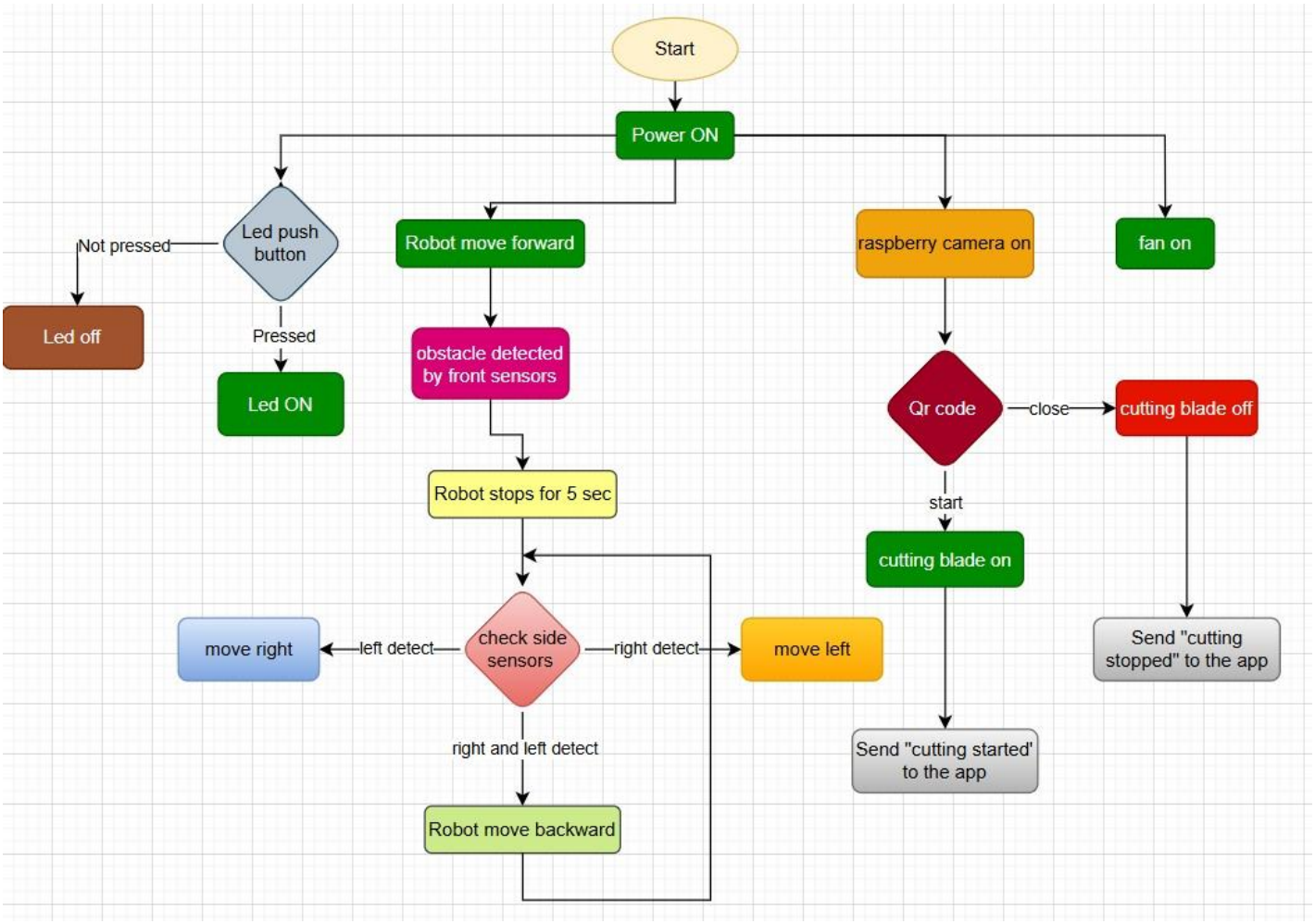
Teffort = $\frac{0.406}{0.065}$ = 6.24 N

TeddortMax = 2.7(9.81)(0.35) = 9.27 N

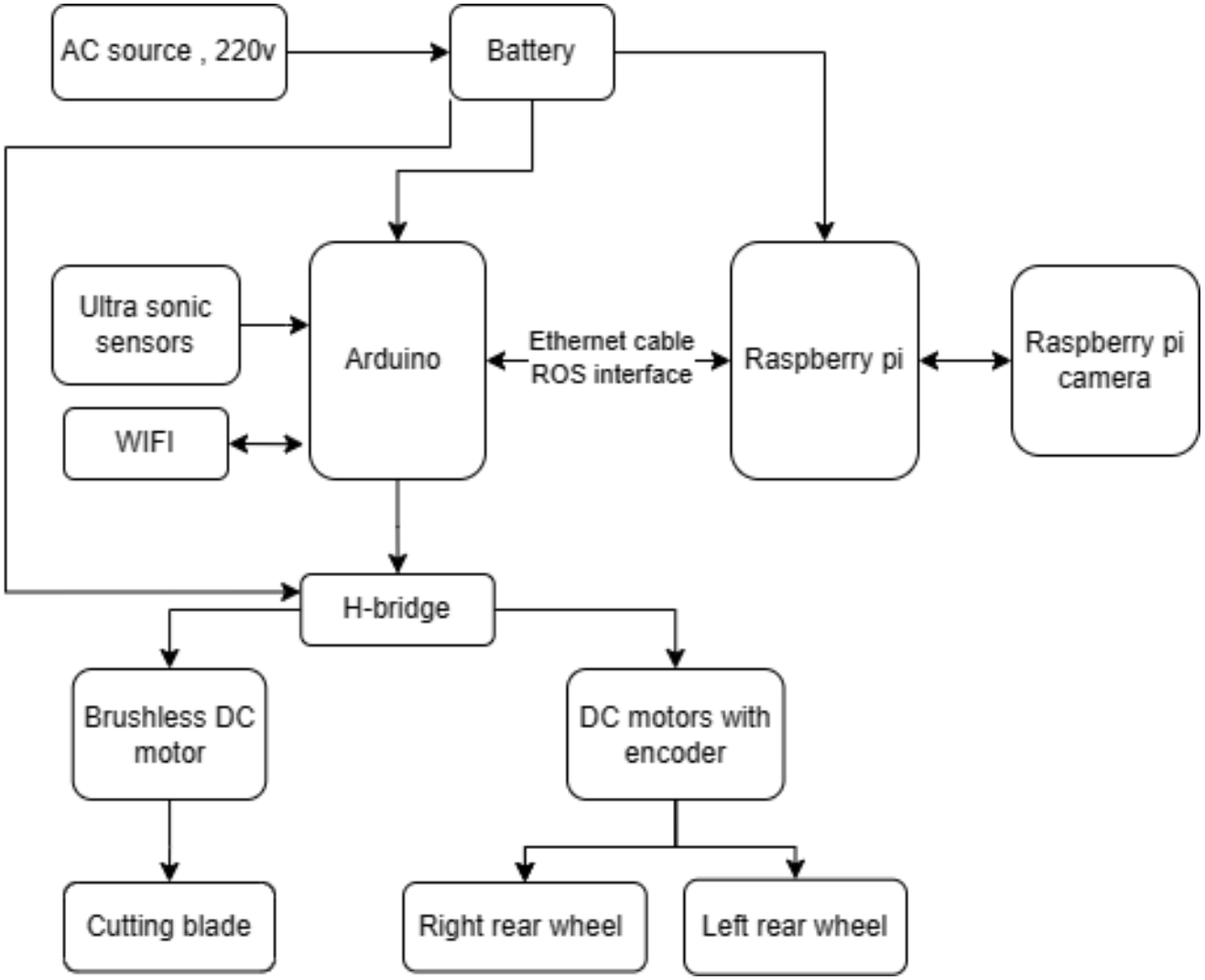
- ✓ Teffort < TeffortMax [design is valid]

Motor selection → DC 12v , 100 rpm , 0.34N.m

8.0. FLOWCHART



9.0. BLOCK DIAGRAM



10.0 SIMULATION

Here we are showing our lawn mower robot simulation by using CoppeliaSim Simulator

10.1. Simulation Environment

The Simulated environment which shows the real environment where our robot operates in which is the garden and there is obstacles in it like garden, people, fountain & buildings.

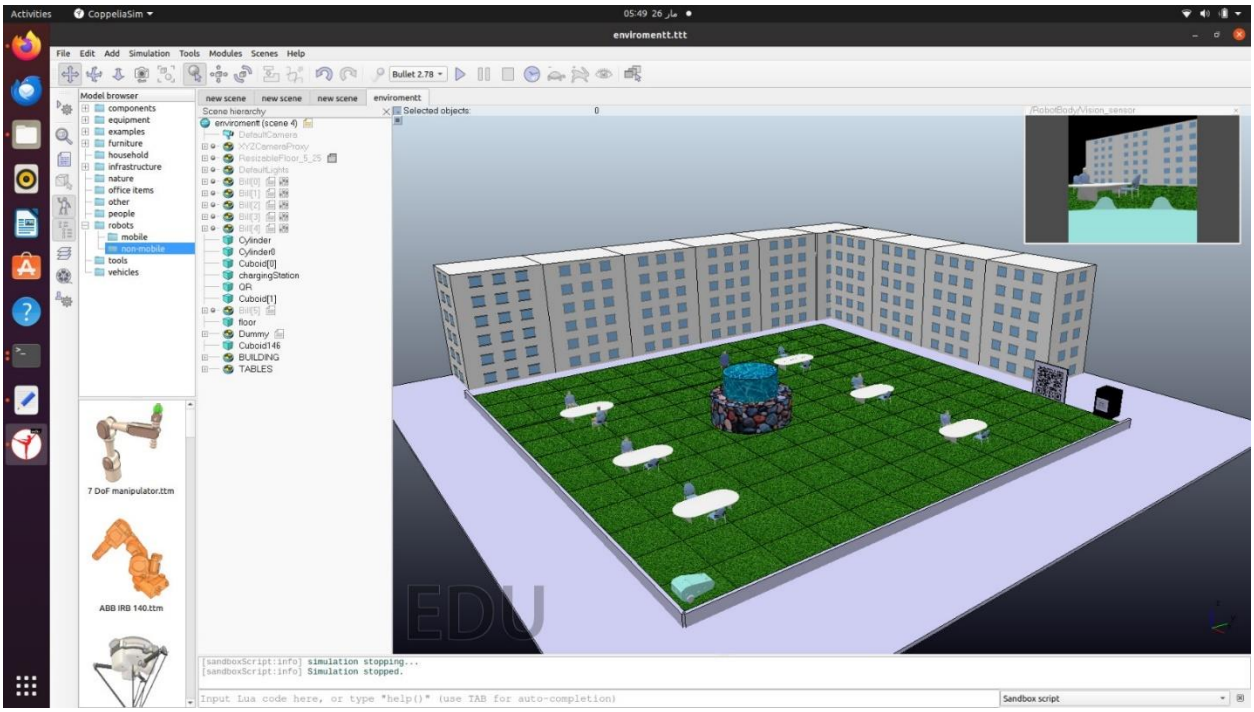


Figure 35: Simulation Environment

10.2. Python/Lua Script with the environment

1. Picture below shows the handler of proximity sensor and cuboid which the robot is on it

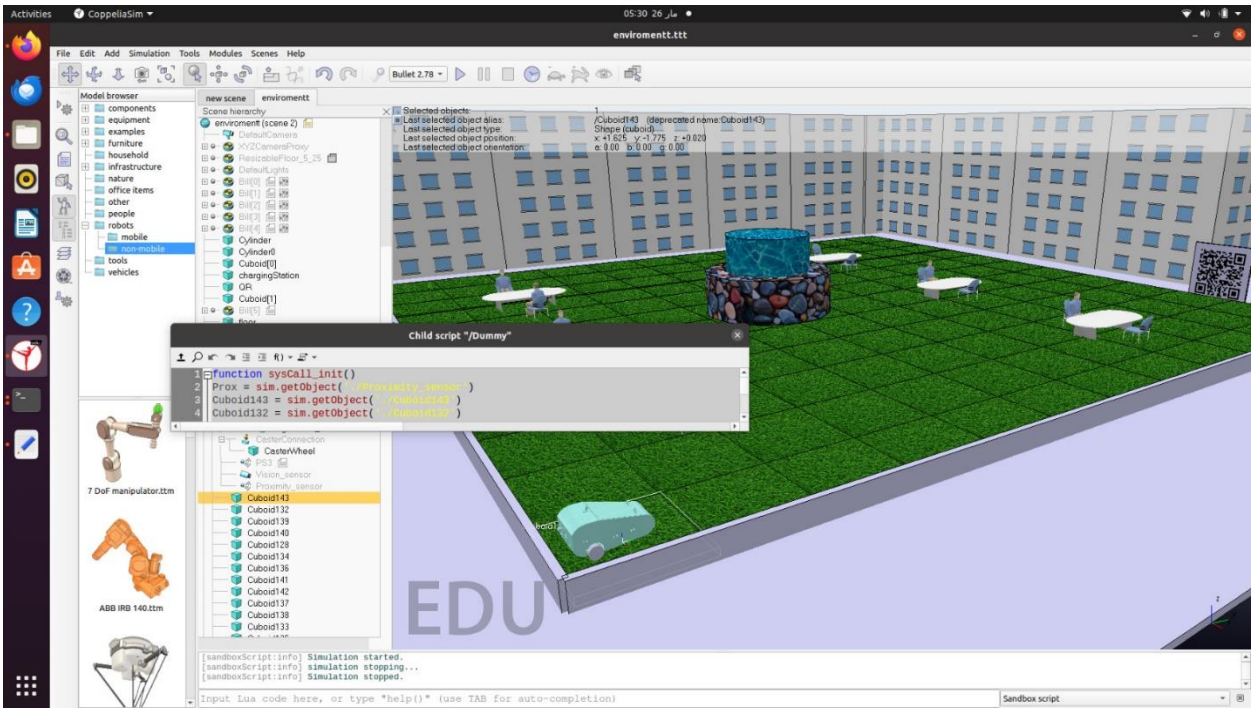


Figure 36: P1

2. this picture shows how the proximity detect the garden under it so the blade can work and cut the grass “decrease its visibility to zero “

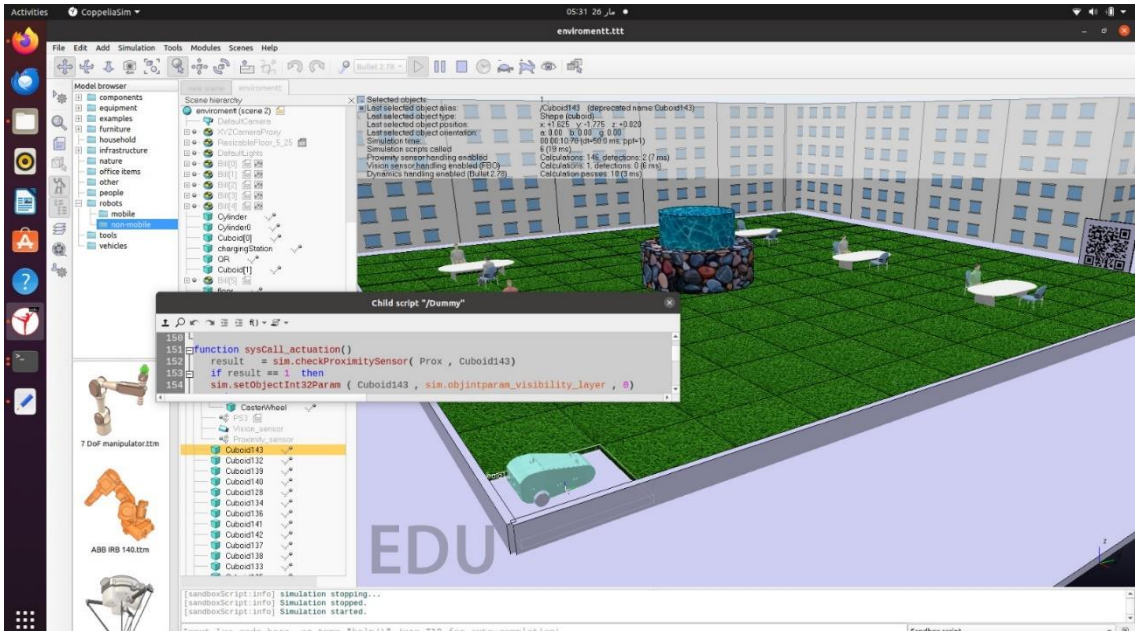


Figure 37: P2

3. This shows sysCall_cleanup which return everything happen during the simulation to its original state when the simulation ends

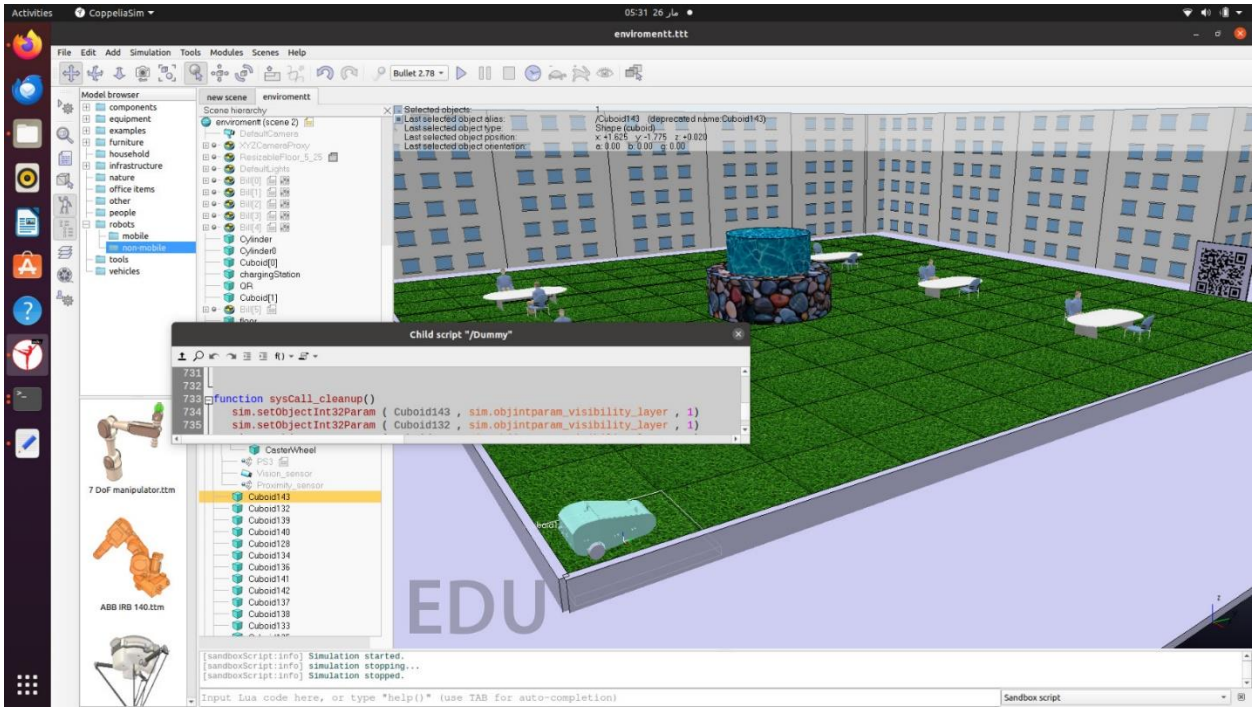


Figure 38: P3

4. At first of while loop it sees the goal point of the trajectory to go to it, if the proximity sensor detect obstacle the robot stops in its place and then check if it is looking to the goal point or not, if not it rotates to the goal point direction and move towards it

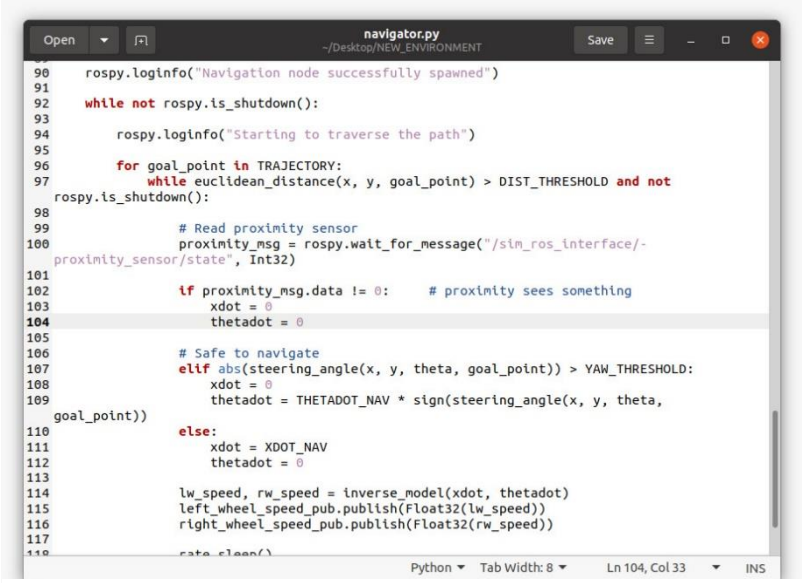


Figure 39:P4

5. Opened interfaces during simulation

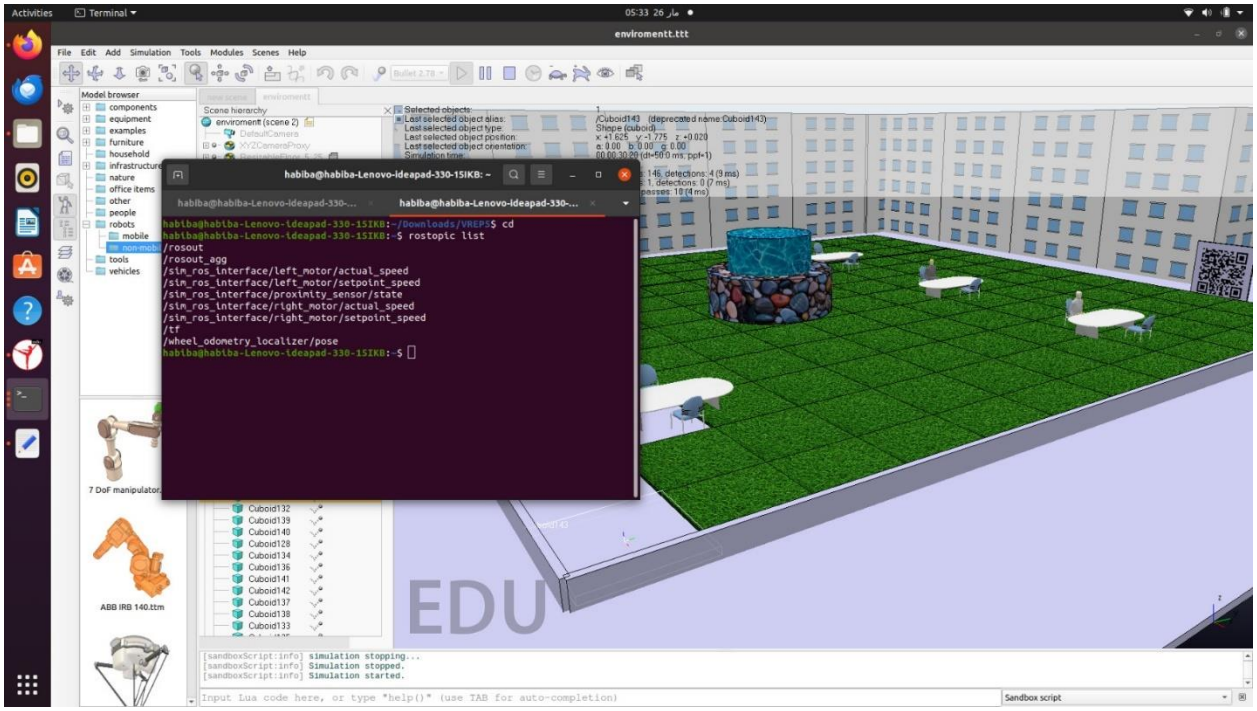


Figure 40: P5

6. This shows the tracing of the proximity sensor results during its detection of object Infront of it

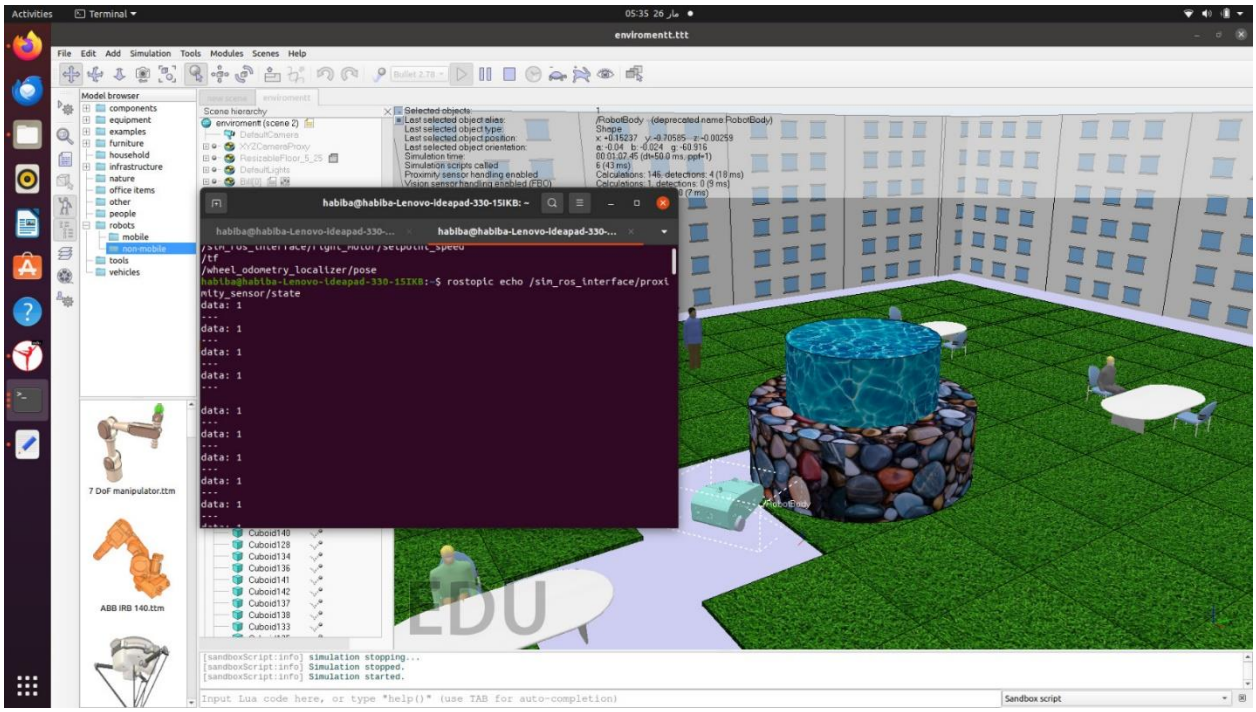


Figure 41: P6

7. This shows the tracing of the proximity sensor results during no object Infront of it

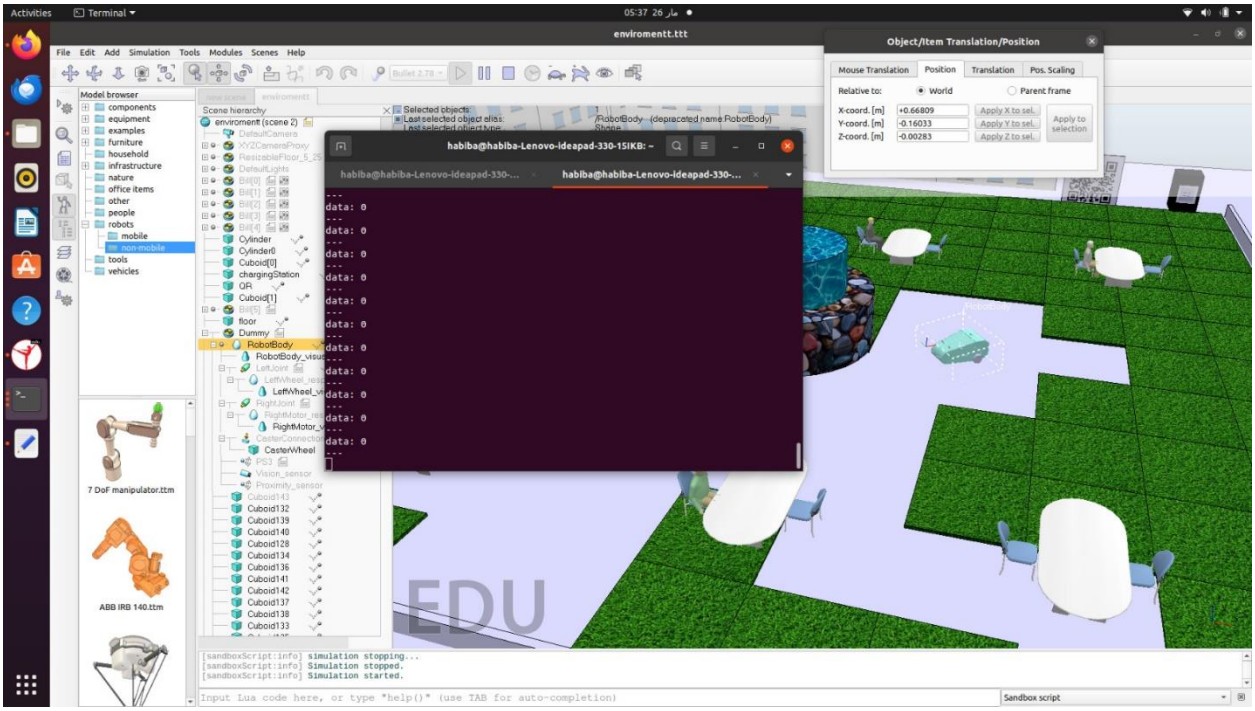


Figure 42:P7

8. Both of the pictures below publish actual speed of joint and subscribe to setpoint speed of joint but one for left motor and one for right motor

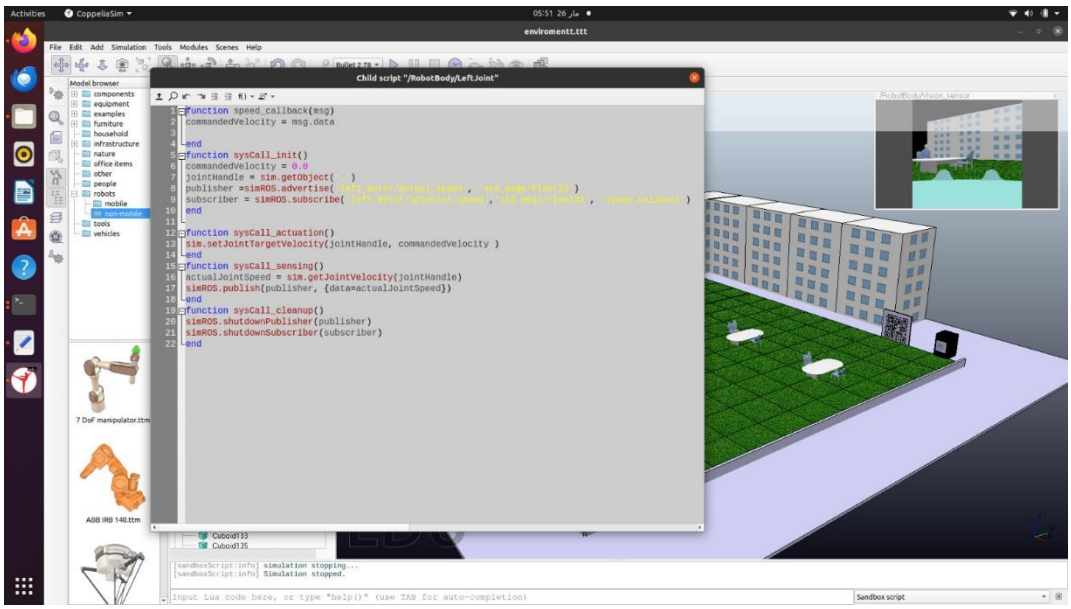


Figure 43: Left Motor Code

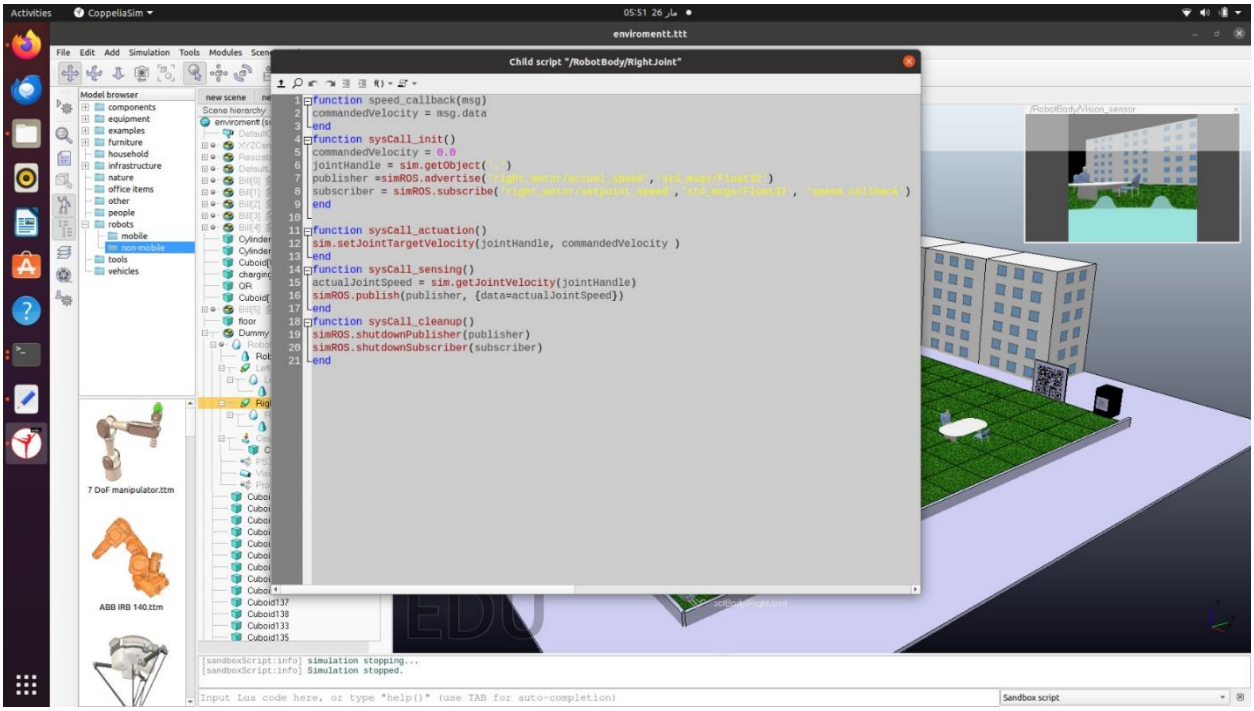


Figure 44: Right Motor Code

9. This shows the proximity how it sends a message by ROS interface which is Boolean 0 if nothing detected and 1 if object detected

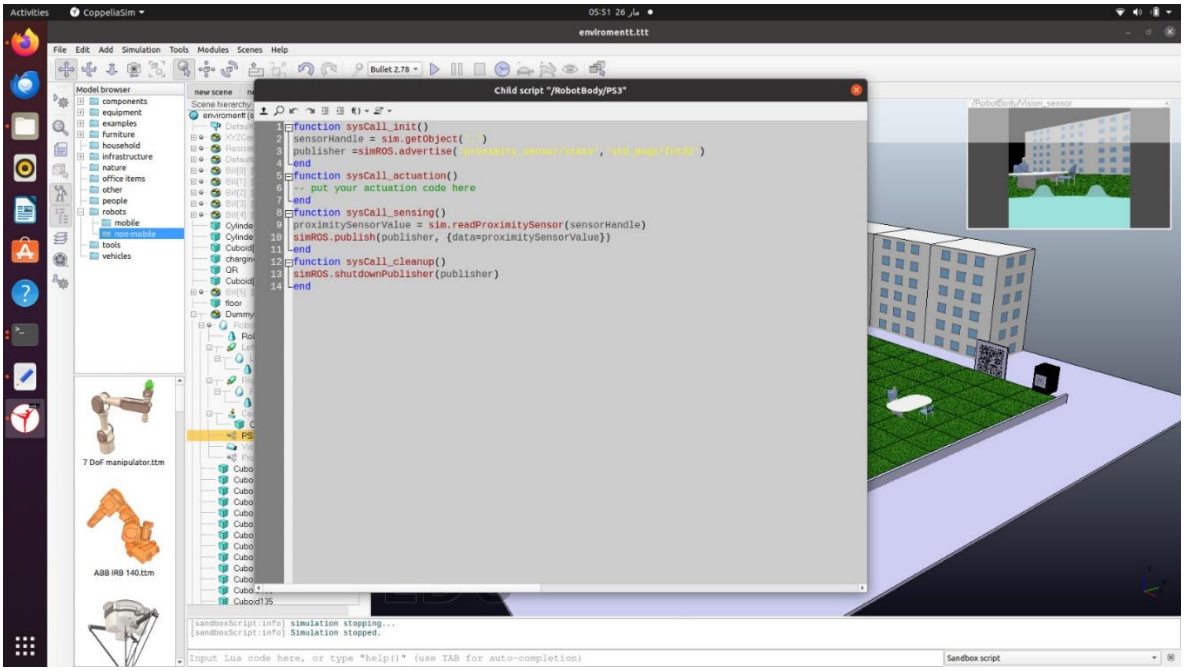


Figure 45: Robot Body Code

11.0 Robot manufactured parts

Here we are showing robot parts in real life

11.1 Robot body frame

The robot's body frame is essential, serving as the foundation for its functions and capabilities, much like the skeletal structure in living organisms. Made of fiber glass, it provides structural support, protection, and a framework for components to operate effectively. Fiber glass offers strength, durability, flexibility, and cost-effectiveness, allowing robots to navigate diverse environments, withstand harsh conditions, and adapt to various applications while maintaining structural integrity and operational efficiency.

Figure 47:Robot body frame1

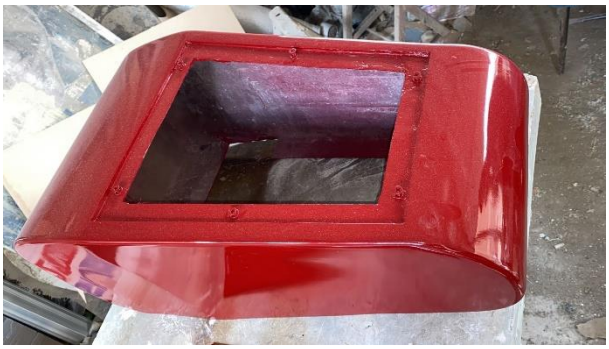


Figure 46:Robot body frame2

11.2 Robot chassis

The sheet metal chassis is essential for robots, providing stability, organization, protection, customization options, weight optimization, and cost-effectiveness.

It offers a sturdy framework for components, shields them from damage, and allows for easy maintenance and upgrades. Additionally, its lightweight nature ensures agility and efficiency in diverse applications, making it a practical choice for both hobbyists and commercial ventures.

the metal chassis serves as a sturdy foundation for the components, but to prevent electrical interference, we've added a layer of cardboard between the components and the metal.

This isolation ensures that the sensitive electronics remain unaffected by any electrical conductivity from the chassis, maintaining optimal performance and reliability of the robot's systems.

Figure 48:Chassis isolated

Within the chassis, we've strategically placed two Arduino Uno boards, an Arduino Mega, a Raspberry Pi, and a custom-designed PCB with multiple ports supporting 12V, 9V, and 5V power sources. This arrangement optimizes space and connectivity, allowing for efficient control and coordination of the robot's functions. the wheels are securely fixed using bearings and shafts, connected to couplers to ensure smooth rotation and minimize friction. Each wheel is directly linked to a motor equipped with an encoder, facilitating precise control and feedback for navigation and motion tracking. This meticulous assembly ensures reliable performance and maneuverability across various terrains and tasks.

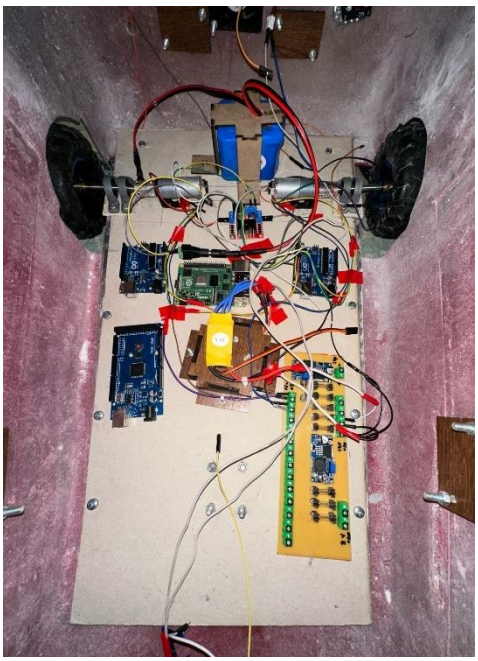


Figure 49:Chassis with components

In our robot, we've securely integrated sensors, switches, and push buttons into the body frame, ensuring they're well-protected and easily accessible. The wheels and various components, including two motors with encoders and one brushless motor, are firmly fixed onto the chassis using bolts and nuts, guaranteeing stability during operation.

Figure 51:Robot pic1

Figure 50:Robot pic2

Figure 53:Robot pic3

Figure 52:Robot pic4

Figure 54:Robot pic5

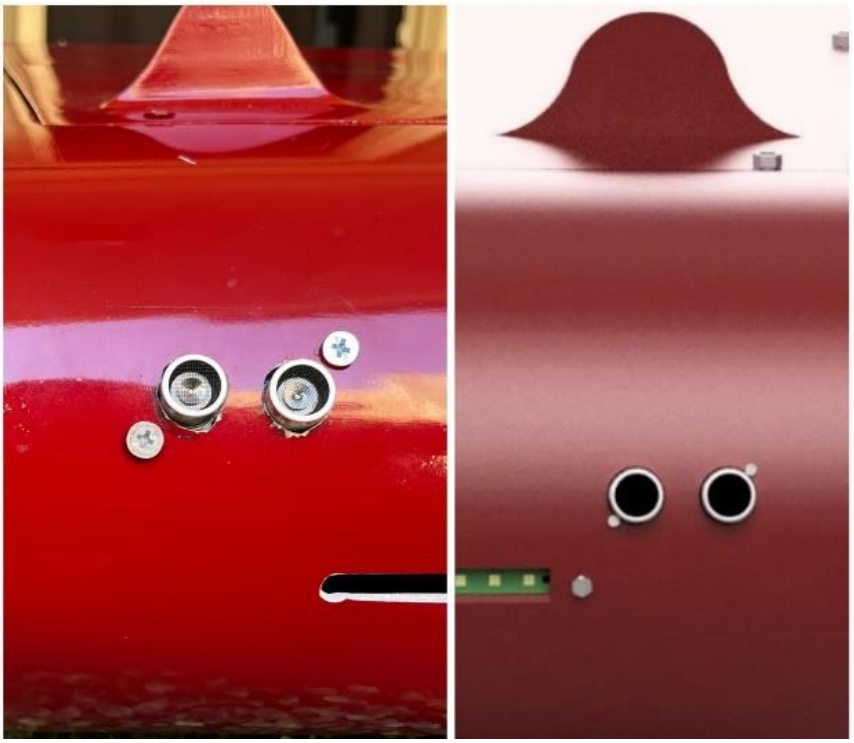


Figure 55:Robot pic6

12.Vision

The whole purpose of the camera in the lawn mower robot project is to control the operation of the cutting blade by scanning QR codes. The Raspberry Pi camera scans these QR codes to activate or deactivate the cutting blade, providing a user-friendly and secure method for managing the blade's operation.

Figure 56: Start Mowing QR code

Figure 57: Stop Mowing QR Code

13.GUI

The whole purpose of the GUI (graphical user interface) in the lawn mower robot project is to provide users with a platform to interact with the robot. Users are required to log in with a username and password, ensuring secure access. The GUI application communicates with the Raspberry Pi to receive real-time status updates of the cutting blade, displaying whether the blade is currently in "Mowing On" or "Mowing Off" mode

Figure 60: GUI Login

Figure 58:p1

Figure 59:p2

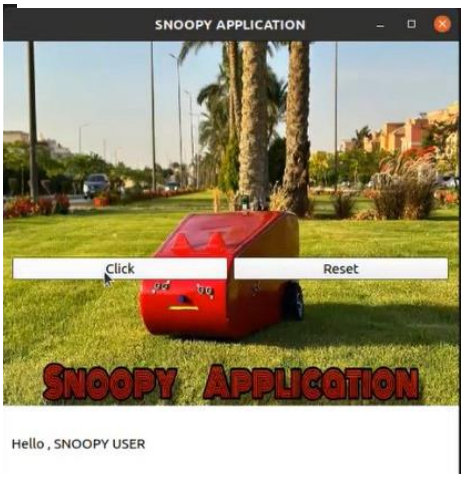


Figure 61:p3

Figure 62:p4

14. 0 Coding

14.1 code

```
// Define the motor right control pins
#define encoderPin1 49
#define encoderPin2 48
#define PWMPin 6
#define in1Pin 53
#define in2Pin 52

// Define the motor left control pins
#define encoderPin1_left 47
#define encoderPin2_left 46
#define PWMPin_left 7
#define in1Pin_left 50
#define in2Pin_left 51

int proportional = 1;
float kpv = 30;
float kiv = 0.11;

int proportional_left = 1;
float kpv_left =5;
float kiv_left = 0.005;

volatile float motorPosition = 0;
volatile float motorPosition_left = 0;
float previousTime = 0;
float previousTime_left = 0;
float errorIntegral = 0;
float errorIntegral_left = 0;
float velocity = 0;
float velocity_left = 0;
float pos = 0;
float pos_left = 0;
float errorvelocity = 0;
float errorvelocity_left = 0;
float controlSignal = 0;
float controlSignal_left = 0;
float controlsignalvelocity = 0;
```

```
float controlsignalvelocity_left = 0;
float target_pos_cm = 140;
float target_pos_cm_left = 140;
float prev_target_pos_cm = 0;
float prev_target_pos_cm_left = 0;
float previousError = 0;
float derivative = 0;
float previousError_left = 0;

// Define the ultrasonic sensor pins
#define trigPin_front1 8
#define echoPin_front1 9

#define trigPin_front2 44
#define echoPin_front2 45

#define trigPin_right 42
#define echoPin_right 43

#define trigPin_left 40
#define echoPin_left 41

long duration_front1, distance_front1;
long duration_front2, distance_front2;
long duration_right, distance_right;
long duration_left, distance_left;

unsigned long lastSensorReadTime = 0;
const unsigned long sensorReadInterval = 50; // Sensor read interval in
milliseconds

void setup() {
  Serial.begin(115200);

  // Motor right setup
  pinMode(encoderPin1, INPUT_PULLUP);
  pinMode(encoderPin2, INPUT_PULLUP);
  pinMode(PWMPin, OUTPUT);
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(encoderPin1), checkEncoder1, RISING);
  attachInterrupt(digitalPinToInterrupt(encoderPin2), checkEncoder2, RISING);

  // Motor left setup
```

```
  pinMode(encoderPin1_left, INPUT_PULLUP);
  pinMode(encoderPin2_left, INPUT_PULLUP);
  pinMode(PWMPin_left, OUTPUT);
  pinMode(in1Pin_left, OUTPUT);
  pinMode(in2Pin_left, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(encoderPin1_left), checkEncoder1_left,
RISING);
  attachInterrupt(digitalPinToInterrupt(encoderPin2_left), checkEncoder2_left,
RISING);

  pinMode(trigPin_front1, OUTPUT);
  pinMode(echoPin_front1, INPUT);

  pinMode(trigPin_front2, OUTPUT);
  pinMode(echoPin_front2, INPUT);

  pinMode(trigPin_right, OUTPUT);
  pinMode(echoPin_right, INPUT);

  pinMode(trigPin_left, OUTPUT);
  pinMode(echoPin_left, INPUT);
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - lastSensorReadTime >= sensorReadInterval) {
    readSensors();
    lastSensorReadTime = currentMillis;
  }

  if (distance_front1 <= 20 || distance_front2 <= 20) {
    stopMotors();
    delay(2000);
   if (distance_front1 <= 20 && distance_front2 <= 20 && distance_right > 20 &&
distance_left > 20 ){
    moveMotors(-50 , -40);
    delay(2000);
    moveMotors(70, -70);
    delay(1500);
   }else {
     moveMotors(50 ,40);
   }

  }
```

```
  else if(distance_right <= 20 && distance_left > 20)
  {
moveMotors(70, -70);
delay(1200);
  }
  else if(distance_left <= 20 && distance_right > 20){
    moveMotors(-70, 70);
    delay(1200);
  }

  else {
    moveMotors(50, 39); // Move forward
  }
}

void readSensors() {
  distance_front1 = getDistance(trigPin_front1, echoPin_front1);
  distance_front2 = getDistance(trigPin_front2, echoPin_front2);
  distance_right = getDistance(trigPin_right, echoPin_right);
  distance_left = getDistance(trigPin_left, echoPin_left);
}

long getDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  return (pulseIn(echoPin, HIGH) * 0.0343) / 2;
}

void moveMotors(int desiredvelocity, int desiredvelocity_left) {
  PID(desiredvelocity, desiredvelocity_left);
}

void stopMotors() {
  PID(0, 0);
}

void PID(int desiredvelocity, int desiredvelocity_left) {
  float currentTime = micros();
  float deltaTime = (currentTime - previousTime) / 1.0e6; // Convert to seconds
  previousTime = currentTime;
```

```
  float currentTime_left = micros();
  float deltaTime_left = (currentTime_left - previousTime_left) / 1.0e6; // Convert to seconds
  previousTime_left = currentTime_left;

  noInterrupts(); // disable interrupts temporarily while reading
  velocity = (motorPosition - pos) / deltaTime; // Corrected calculation
  velocity_left = (motorPosition_left - pos_left) / deltaTime_left; // Corrected calculation
  pos = motorPosition;
  pos_left = motorPosition_left;
  interrupts(); // turn interrupts back on

  float desiredpos = target_pos_cm;
  controlSignal = (desiredpos - pos) * proportional;

  float desiredpos_left = target_pos_cm_left;
  controlSignal_left = (desiredpos_left - pos_left) * proportional_left;

  // Limit control signals to desired velocity
  controlSignal = constrain(controlSignal, -desiredvelocity, desiredvelocity);
  controlSignal_left = constrain(controlSignal_left, -desiredvelocity_left, desiredvelocity_left);

  // Calculate errors and PID terms
  errorvelocity = desiredvelocity - velocity;
  errorIntegral += errorvelocity * deltaTime;
  derivative = (errorvelocity - previousError) / deltaTime;
  previousError = errorvelocity;

  errorvelocity_left = desiredvelocity_left - velocity_left;
  errorIntegral_left += errorvelocity_left * deltaTime_left;
  float derivative_left = (errorvelocity_left - previousError_left) / deltaTime_left; // Declare derivative_left locally
  previousError_left = errorvelocity_left;

  // Calculate control signals with PID terms
  controlsignalvelocity = kpv * errorvelocity + kiv * errorIntegral; // Added kdv term
  controlsignalvelocity_left = kpv_left * errorvelocity_left + kiv_left * errorIntegral_left; // Used local derivative_left

  // Convert control signals to PWM values
  int motorDirection = (controlsignalvelocity >= 0) ? 1 : -1;
```

```
  int motorDirection_left = (controlsignalvelocity_left >= 0) ? 1 : -1;
  float PWMValue = constrain(fabs(controlsignalvelocity), 0, 255);
  float PWMValue_left = constrain(fabs(controlsignalvelocity_left), 0, 255);

  // Set motor directions and PWM values
  analogWrite(PWMPin, PWMValue);
  analogWrite(PWMPin_left, PWMValue_left);
  digitalWrite(in1Pin, motorDirection == 1 ? HIGH : LOW);
  digitalWrite(in2Pin, motorDirection == -1 ? HIGH : LOW);
  digitalWrite(in1Pin_left, motorDirection_left == 1 ? HIGH : LOW);
  digitalWrite(in2Pin_left, motorDirection_left == -1 ? HIGH : LOW);
}

void checkEncoder1() {
  motorPosition++;
}

void checkEncoder2() {
  motorPosition--;
}

void checkEncoder1_left() {
  motorPosition_left++;
}

void checkEncoder2_left() {
  motorPosition_left--;
}
```

14.2 Code explanation

This Arduino code is for controlling a robot with two motors and ultrasonic sensors. Here's a breakdown of its key components and functionality:

Definitions and Declarations

Motor Control Pins

- **Right Motor:**
  - **`encoderPin1`**, **`encoderPin2`**: Pins for the right motor encoder.
  - **`PWMPin`**: PWM pin for controlling speed.
  - **`in1Pin`**, **`in2Pin`**: Pins for setting the motor direction.
- **Left Motor:**
  - **`encoderPin1_left`**, **`encoderPin2_left`**: Pins for the left motor encoder.
  - **`PWMPin_left`**: PWM pin for controlling speed.
  - **`in1Pin_left`**, **`in2Pin_left`**: Pins for setting the motor direction.

PID Control Variables

- **Right Motor:**
  - **`kpv`**, **`kiv`**: PID control parameters for velocity.
  - Variables to store motor position, velocity, error, and control signals.
- **Left Motor:**
  - **`kpv_left`**, **`kiv_left`**: PID control parameters for velocity.
  - Similar variables as the right motor for position, velocity, error, and control signals.

Ultrasonic Sensor Pins

- **Sensors:**
  - `trigPin_front1`, `echoPin_front1`: Pins for the front left sensor.
  - `trigPin_front2`, `echoPin_front2`: Pins for the front right sensor.
  - `trigPin_right`, `echoPin_right`: Pins for the right side sensor.
  - `trigPin_left`, `echoPin_left`: Pins for the left side sensor.

Sensor Reading Variables

- Variables to store the duration and distance measurements from the ultrasonic sensors.
- `lastSensorReadTime`, `sensorReadInterval`: To manage sensor reading intervals.

Setup Function

- Initializes serial communication.
- Sets the pin modes for motor control and ultrasonic sensors.
- Attaches interrupts for the encoder pins to track motor positions.

Loop Function

- Continuously checks the time and reads sensor values at regular intervals.
- Based on sensor readings, decides the robot's movement:
  - Stops and adjusts direction if an obstacle is detected within 20 cm in front or on the sides.
  - Moves forward if the path is clear.

Sensor Reading Function (`readSensors`)

- Measures distances using the ultrasonic sensors by triggering the sensors and calculating the time it takes for the echo to return.

Distance Calculation Function (`getDistance`)

- Sends a pulse and calculates the distance based on the time taken for the echo to return.

Motor Control Functions

- **Move Motors (`moveMotors`)**: Calls the PID function to adjust motor speeds.
- **Stop Motors (`stopMotors`)**: Stops the motors by setting the desired velocity to zero.

PID Control Function (`PID`)

- Calculates the time delta for velocity calculations.
- Computes the current motor velocity based on encoder positions.
- Calculates control signals using PID control (Proportional, Integral, Derivative terms).
- Adjusts motor PWM values and directions based on the control signals.

Encoder Interrupt Service Routines

- **Right Motor:**
  - **`checkEncoder1`**, **`checkEncoder2`**: Updates **`motorPosition`** based on encoder signals.
- **Left Motor:**
  - **`checkEncoder1_left`**, **`checkEncoder2_left`**: Updates **`motorPosition_left`** based on encoder signals.

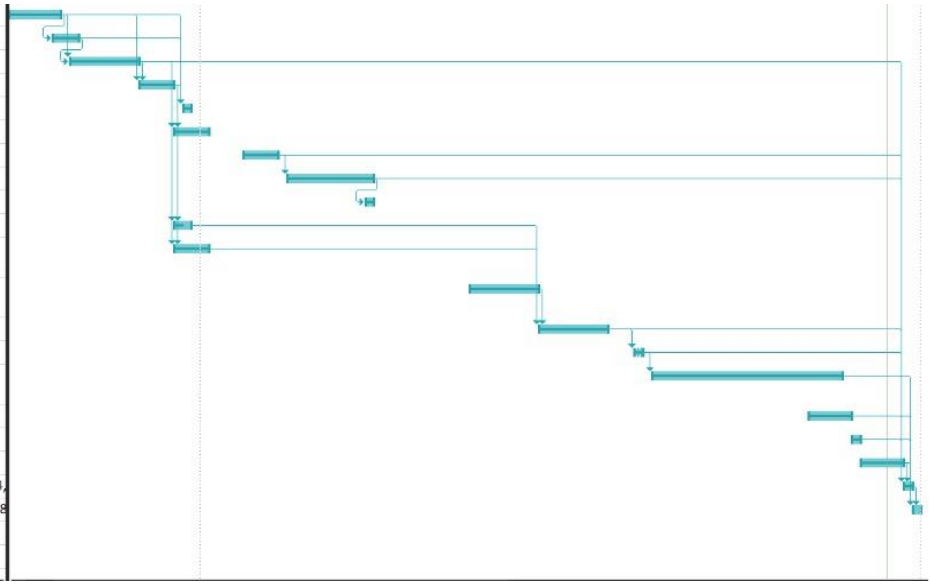
This code enables a robot to move, avoid obstacles using ultrasonic sensors, and control motor speeds using PID control.

15.0 Robot Cost

Components	Cost
Bolts	50 EGP
On/Off Switch	50 EGP
Cooling Fan	60 EGP
H Bridge	85 EGP
Emergency Switch	100 EGP
Couplers	120 EGP
Bearings	200 EGP
Raspberry Pi Camera	200 EGP
Chassis	280 EGP
Wheels	300 EGP
ESC Driver	300 EGP
Ultrasonic Sensors	300 EGP
Microcontroller PCB	500 EGP
Brushless DC Motor	560 EGP
Battery	600 EGP
Arduino Mega	600 EGP
Dc Motors	1400 EGP
Body Frame	2000 EGP
Raspberry pi 4 8GB	6,500 EGP
Total Cost	**14,205 EGP**

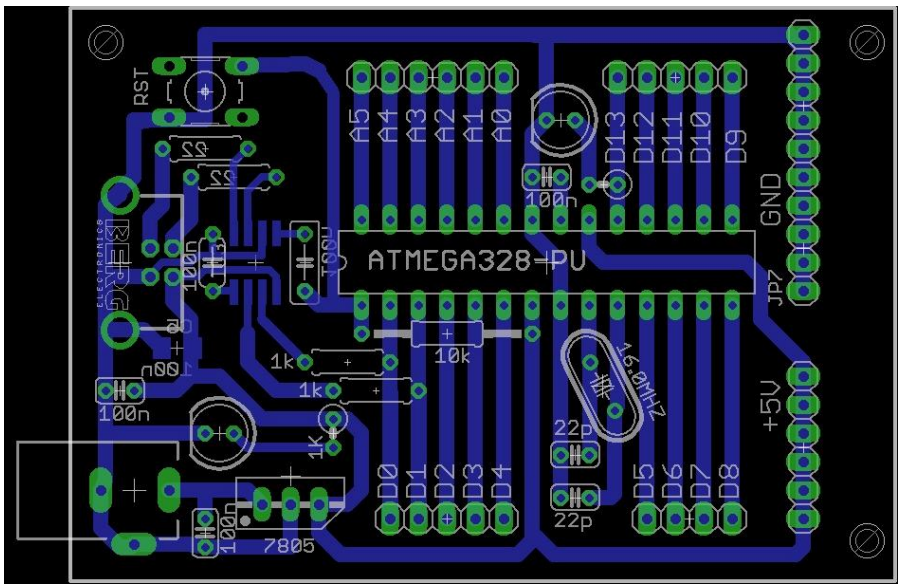
16.0 Chart Timeline

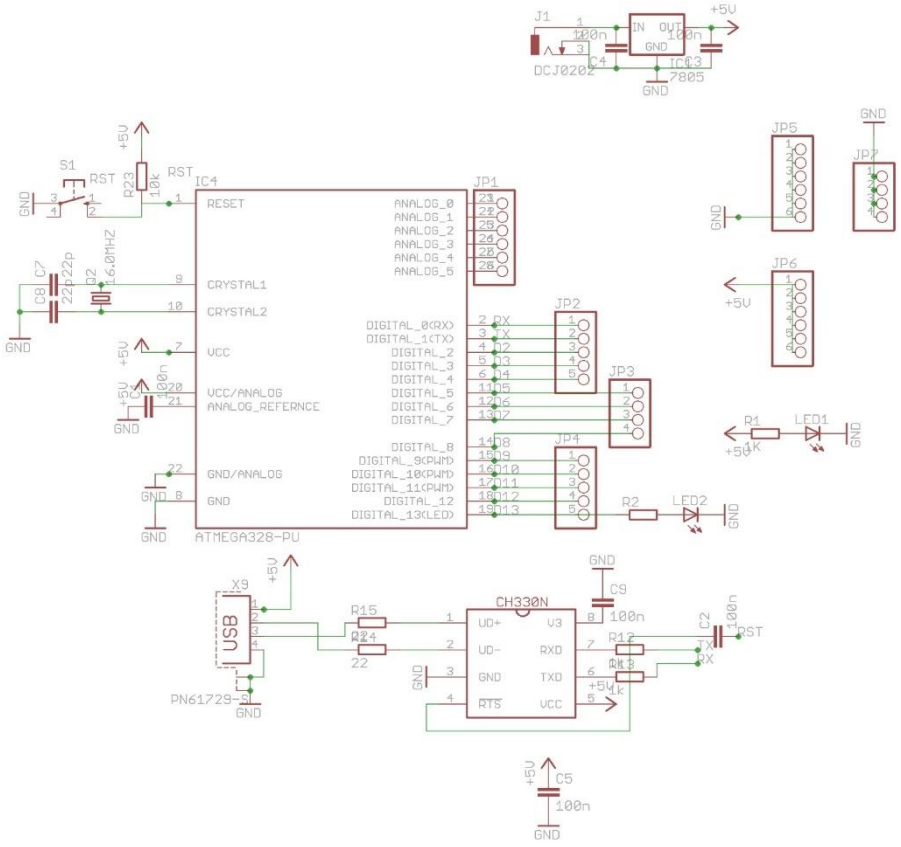
📌	Define Robot Function	4 days	Thu 2/15/24	Tue 2/20/24	
📌	Sketch design	3 days	Tue 2/20/24	Thu 2/22/24	1
📌	CAD desgin	6 days	Thu 2/22/24	Thu 2/29/24	1,2
📌	final assembly	2 days	Fri 3/1/24	Mon 3/4/24	1,2,3
📌	Quiz(1)	1 day	Wed 3/6/24	Wed 3/6/24	1,2,3,4
📌	actuator sizing	4 days	Tue 3/5/24	Fri 3/8/24	3,4
📌	Prepare enviroment	4 days	Wed 3/13/24	Sat 3/16/24	
📌	robot simulation	8 days	Mon 3/18/24	Wed 3/27/24	7
📌	Midterm submission	1 day	Wed 3/27/24	Wed 3/27/24	8
📌	Chassis manufacturing	2 days	Tue 3/5/24	Wed 3/6/24	3,4
📌	body frame manufacturing	4 days	Tue 3/5/24	Fri 3/8/24	3,4
📌	buy electrical components	6 days	Mon 4/8/24	Mon 4/15/24	
📌	integration	6 days	Tue 4/16/24	Tue 4/23/24	10,11,12
📌	Major task submission	1 day	Sat 4/27/24	Sat 4/27/24	13
📌	PID controller with sensors	16 days	Mon 4/29/24	Mon 5/20/24	14
📌	vision	3 days	Fri 5/17/24	Tue 5/21/24	
📌	microcontroller pcb	1 day	Wed 5/22/24	Wed 5/22/24	
📌	GUI	3 days	Thu 5/23/24	Mon 5/27/24	
📌	poster	1 day	Tue 5/28/24	Tue 5/28/24	3,7,8,13,14,
📌	final submission	1 day	Wed 5/29/24	Wed 5/29/24	15,16,17,18



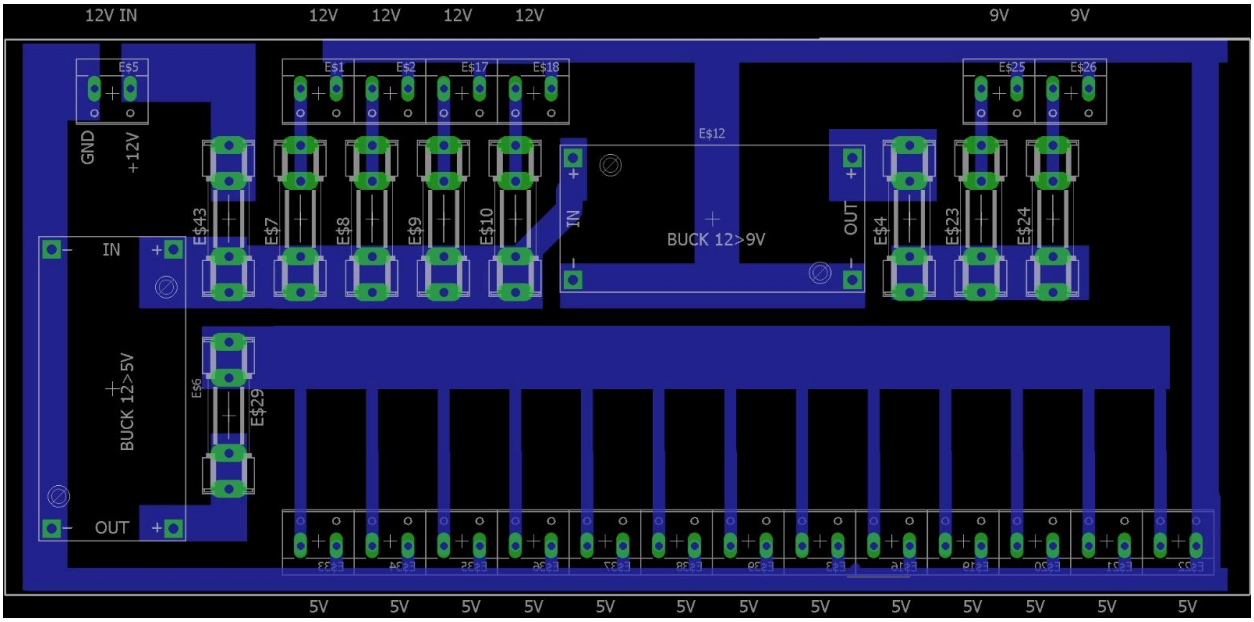
17.0 Circuit Integration

17.1 Arduino uno pcb

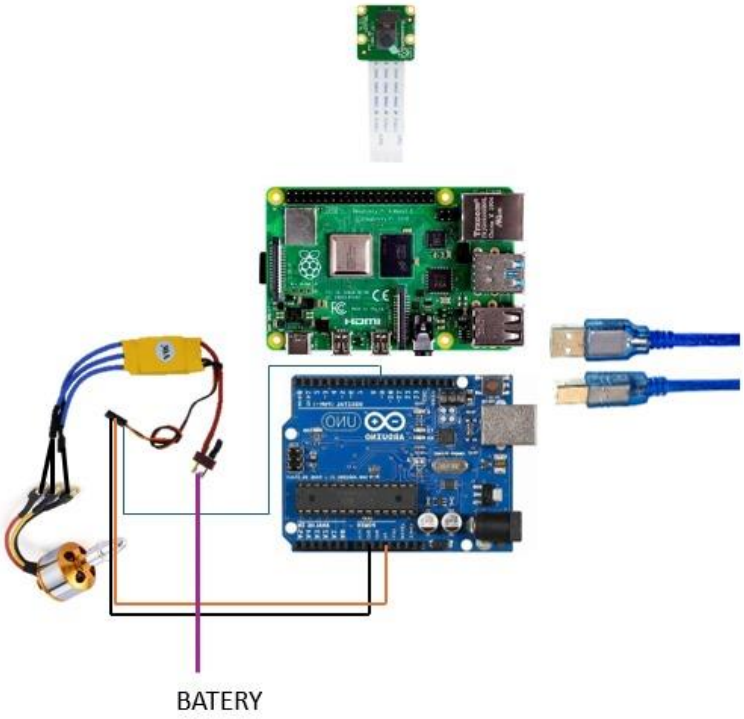


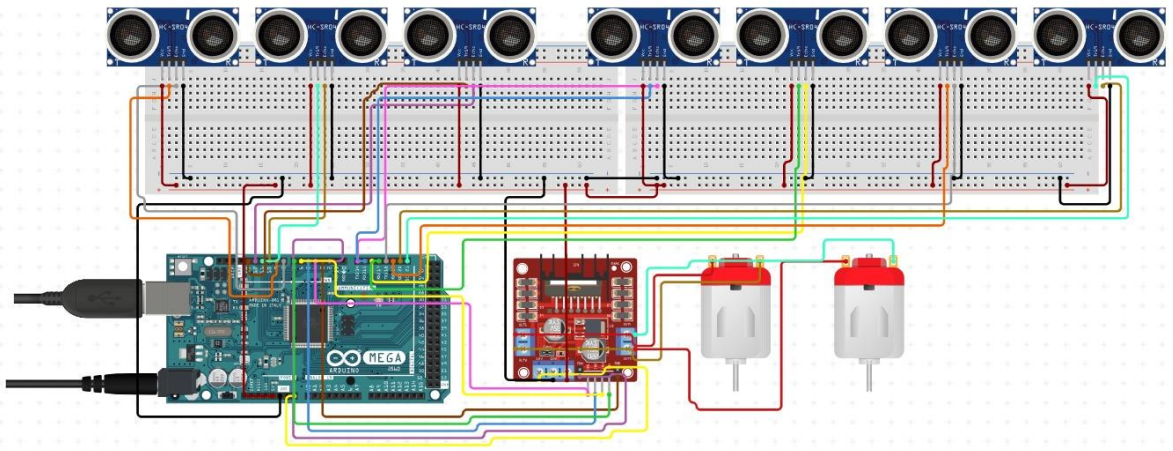


17.2 Power Supply pcb



17.3 Circuit Wiring





18.0 DRIVE LINK

https://drive.google.com/drive/folders/14DIo1OblmqT264qOgLTCd4sbxr6UEtY9?usp=drive_link