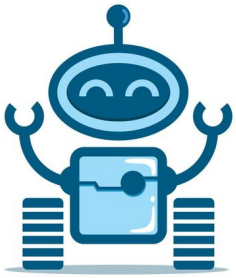
An Najah National University

Faculty of Engineering & Technology
Computer Engineering Department

Hardware Graduation Project

WALL-E Savior Robot



Prepared By: Marah Akef Hanini and Ahmad Mohammed Hussein

Supervised By: Dr. Muhannad al-jabi

Presented in partial fulfillment of the requirements for a Bachelor's degree in Computer Engineering

January, 2025

Acknowledgment

First and foremost, we want to express our heartfelt gratitude to our parents and friends for their unwavering support throughout our academic journey. Your encouragement has been the driving force behind our determination, and this achievement is as much yours as it is ours.

We extend a special thanks to our supervisor, Dr. muhanned al-jabi , for his invaluable guidance and dedication. His expertise has been instrumental in shaping our project and bringing it tofruition.

We are also grateful to the faculty members of the College of Engineering and the Computer and Electrical Engineering Department. Your knowledge and guidance have been the cornerstone of our success, and we deeply appreciate your contributions.

Finally, to our classmates and friends, thank you for the companionship and the shared experiences that made this journey truly memorable.

With sincere appreciation,

Marah hanini
Ahmad hussien

Disclaimer

This report was created by Marah hanini and Ahmad hussien from the Computer Engineering Department at the Faculty of Engineering, An-Najah National University.
Please note that it hasn't been revised beyond minor editorial fixes, so it might still have some language or content errors.
The opinions, outcomes, and recommendations in this report are entirely those of the authors. An-Najah National University is not responsible for any consequences if this report is used for anything other than its intended purpose.

Table of Contents

List of Figures

Abstract

Our project is about developing a robot that is to be used in hazardous environments where it is dangerous for humans to approach. The robot's main objective is to help in performing crucial tasks, as extinguishing fires or detecting dangerous objects, especially when it comes to Effectively neutralize dangerous objects. Using digital image processing the robot is able to identify wire colors in order to help the human expert identify which wire to cut.

The robot features with a sturdy and rust-resistant body made of galvanized steel, enabling it to operate effectively under harsh conditions. Its design is inspired by the character Wall-E, adapted to meet the requirements of its specialized tasks. With dimensions that exceeding one meter in height , the robot is equipped with high-capacity motors that are capable of handling significant weights.

rechargeable lithium battery powers the robot, which is custom designed to provide the ideal voltage and current capacity for continuous use. The precision movements of the robotic arms are made by stepper motors and data exchange and control are through a combination of Raspberry Pi, Arduino Mega . The video feed from the onboard camera is used for both digital image processing and for human controlled operations as visual input. This robot is an innovative new prototype to improve safety and efficiency in hazardous environments.

Chapter 1

Introduction

In various high-risk environments, such as high-pressure areas and hazardous zones, performing tasks like cutting or connecting wires presents significant challenges and risks to human safety. Traditional approaches often require manual intervention, which can be dangerous and inefficient. Technological advancements in robotics and digital image processing provide innovative solutions to mitigate these risks and enhance operational efficiency.

1.1 Objectives

develop an advanced robot capable of safely and effectively operating in hazardous environments, focusing on the requirements of safety, precision and efficiency in high risk conditions. This robot is specifically designed to reduce the risks that human experts have to face and in doing so to protect them from danger by performing tasks in areas that are too dangerous for people to enter directly. It will be developed with fire detection and suppression system which uses fire sensor and extinguisher to detect and put out fire quickly to minimize damage and prevent expansion of a hazardous situation.

In addition, the robot will have a wireless remote control system which allows human experts to control the robot from a safe position. This functionality is not only safe but also gives the operators high level of control over the robot's movement and actions which is very important for the successful completion of critical tasks. In order to support extended and demanding operation, the robot will be designed to run on high capacity rechargeable batteries. This ensures that the robot can function properly and without interruption, and it is able to last for long periods of time, even in the most difficult conditions. In the process, our project seeks to develop a smart and multipurpose robotic system that can help solve a variety of problems in hazardous environments in order to enhance safety and performance.

1.2 Significance

The goal of our project, WALL-E Savior Robot, is to improve greatly the safety of humans by authorizing the remote execution of dangerous and high risk tasks such as putting out fires or identifying dangerous objects. This makes direct human interaction with hazardous environments unnecessary hence minimizing the chances of getting hurt or getting involved in an accident.

With the use of image processing technology and robotics, our project is the most advanced in the field and is a clear example of how new technologies can be used to solve problems in safety and emergency management. The robot is precise and reliable in its operations and is able to work well in the most difficult conditions.In addition, the system is a trustworthy and effective asset for emergency personnel, allowing them to deal with potential challenges such as bomb disposal and fire fighting in a more controlled and safer manner. This innovative approach cannot be transformed into this project that has the capability of saving lives and increasing the efficiency of operations in hazardous conditions.

Chapter 2

Theoretical Background and Previous Work

2.1 Historical Context

The evolution of robotics has been deeply intertwined with humanity's quest to enhance safety, efficiency, and precision in hazardous environments. Early robotic systems were primarily designed for industrial automation, focusing on repetitive manufacturing tasks. Over time, advancements in technology paved the way for more specialized robots capable of operating in challenging and high-risk scenarios. A significant milestone in this evolution was the development of remotely operated robots during the late 20th century, particularly for military applications like reconnaissance.

These early robots demonstrated the potential of wireless communication and precise manipulators in reducing human exposure to danger.The historical progression of robotics, combined with advancements in image processing and wireless communication, has set the stage for innovative systems like the WALL-E Savior Robot. Inspired by decades of research and development, this project builds on proven technologies to address modern challenges, offering a versatile solution for fire suppression, wire handling,
and remote operations in dangerous environments. By integrating past innovations with contemporary needs, the project represents a continuation of robotics' journey toward enhancing safety and efficiency in critical scenarios.

2.2 Existing Literature

Wireless technologies like Wi-Fi, Bluetooth, and RF modules (e.g., NRF24) are fundamental in ensuring real-time communication between robots and operators. Wi-Fi offers high-speed data transfer, making it ideal for transmitting live video feeds and sensor data.
Bluetooth is efficient for short-range, low-energy communication, often used in compact robotic systems. RF modules like the NRF24 provide low-latency, long-range connectivity, ensuring reliable control even in complex environments.
These technologies enable smooth interaction and control, which are crucial for precision tasks,

especially in hazardous scenarios where human safety is a priority .[2]

Fire-Fighting Robots: Robots equipped with fire extinguishers and flame detection sensors are designed to address fire-related emergencies in industrial and residential settings.
For instance, the Firefighting Assistant Robot (FAR) uses flame sensors to detect fires and suppression systems to extinguish them. These robots can navigate autonomously or be controlled remotely, minimizing human risk in dangerous environments.
They are particularly useful in high-risk areas with hazardous chemicals, enabling effective fire containment and improving safety standards.[1]

Chapter 3

Methodology

In this chapter, we provide a detailed explanation of the materials, methods, and standards employed in the development of the WALL-E Savior Robot. The methodology outlines the design and integration of various components, each playing a critical role in ensuring the robot's ability to safely and efficiently operate in hazardous environments. From fire detection and suppression to precision wire handling and remote operation, each aspect of the robot has been carefully designed and implemented to address the challenges associated with high-risk tasks.

3.1 Standards and Specifications

Our design adheres to engineering standards to ensure reliability and compatibility. Notably, the IEEE 802.11 standard is employed for communication protocols, focusing on seamless connectivity and efficient data exchange within the system. Additionally, the design rigorously complies with safety standards, enhancing overall safety and user well-being.

3.2 Hardware Components

3.2.1 Microcontrollers and Processors

- Arduino Mega 2560: The Arduino Mega 2560 acts as the main control unit for the entire fire system. It is connected to all critical components, including sensors, lights, display, motors, and actuators. With its large number of input/output pins, it efficiently manages the signals and operations, coordinating all system functions to detect and extinguish fires.

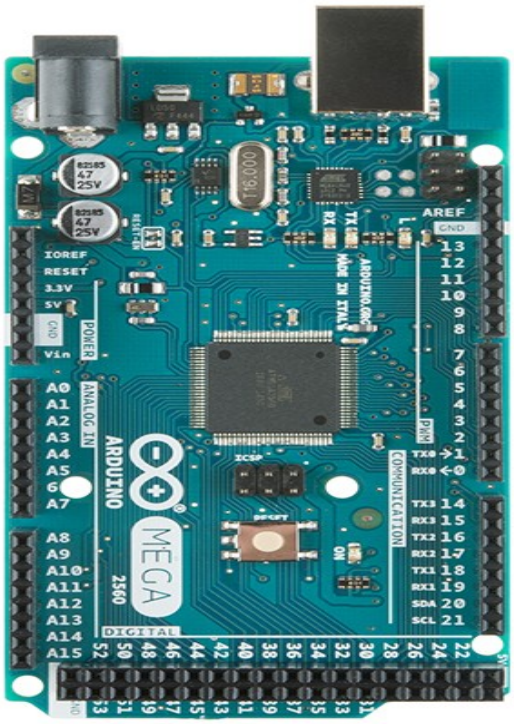


Figure 3.1: Arduino Mega 2560

- ESP32:we use it solely for programming the robot's eyes enabling dynamic visual expressions and interactions.

Figure 3.2: ESP32

- Arduino Uno: This board controls the fire system, the GPS module, and the arm movements of the robot. Its simple yet effective design guarantees accurate placement, precise arm control, and dependable fire suppression mechanism operation.

Figure 3.3: Arduino Uno

- Raspberry Pi 5: With a multi-core processor and sophisticated networking features, the Raspberry Pi 5 is a powerful single-board computer. Accurate object identification and analysis were made possible by its use in this project for image processing tasks like color detection and camera integration.

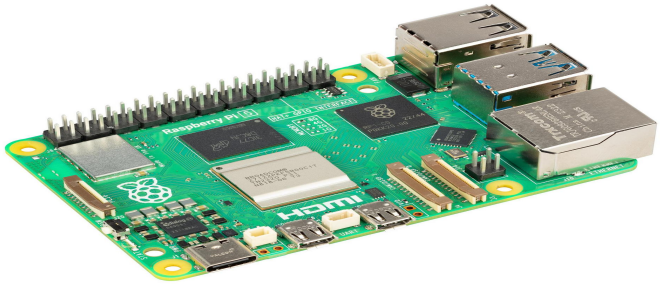


Figure 3.4: Raspberry Pi 5

3.2.2 Motors and Drivers

- **Wiper Motor:** Made for automotive use, the wiper motor is a high-torque DC motor. Its tremendous torque capabilities make it appropriate for supporting the robot's weight and guaranteeing steady and potent locomotion, which is why it is used in this project.

Figure 3.5: Wiper Motor

- **Drill Motor:** The drill motor is used to operate the fire extinguisher. Once the sensor detects fire and sends a signal, the servo motor activates the drill motor, which performs four rotations to engage the extinguisher. This ensures precise activation and deactivation of the fire suppression system.

Figure 3.6: Drill Motor

- **Stepper Motor :** Stepper motors are used in this project to provide precise and controlled movements for the robot's arm. They ensure the arm can perform tasks such as gripping, lifting, and manipulating objects with high precision and stability. The stepper motors' ability to move in small, defined increments allows for fine control, making them ideal for tasks that require exact movements and handling of different objects.

Figure 3.7: Stepper Motor

- **Servo Motor:** The first arm's motion toward the fire extinguisher nozzle is managed by the servo motor. The servo motor rotates the arm to position the nozzle, turning on the fire extinguisher, when the temperature sensor detects heat at a particular angle. To ensure that the arm resets to its initial condition after the fire is put out, the servo motor rotates the same number of turns in reverse to return the arm to its starting position.

Figure 3.8: Servo Motor

- **DC Motor Driver:** The DC motor driver controls the speed and direction of the DC motors, enabling smooth and responsive movement for the robot.

Figure 3.9: DC Motor Driver

- **Stepper Motor Driver:** The stepper motor driver precisely controls the position, speed, and direction of the stepper motor, enabling accurate and reliable movement for the robot in applications requiring high precision.

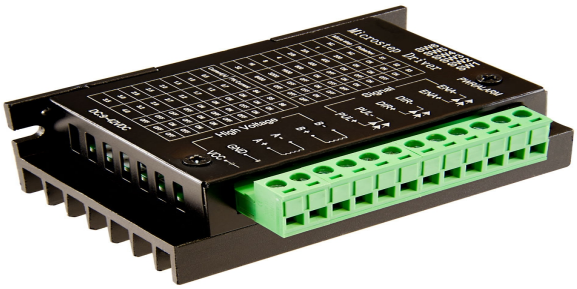


Figure 3.10: Stepper motor Driver

3.2.3 Sensors

- **fire sensor:** Heat or flames are detected by the fire sensor. When a fire is detected, it causes the system to engage the firefighting mechanism, enabling the robot to react to fire dangers in a timely and effective manner.

Figure 3.11: fire sensor

3.2.4 Actuators and Mechanical Components

- **Bicycle Chain:** The robot's wheels are connected to and powered by the bicycle chain. By ensuring effective torque transfer from the engine to the wheels, it enables the robot to move steadily and manage big loads.

Figure 3.12: Bicycle Chain

- **Bicycle gears:** Power is transferred from the motor to the robot's tank-like legs via the bicycle gears and the bicycle chain. They provide the torque and control required for negotiating challenging terrain and preserving stability in dangerous situations, ensuring efficient and smooth mobility.

Figure 3.13: Bicycle Chain

- **Clamp Hand:** This hand is employed to grasp and control items. During operations, the robot can safely handle a variety of things, including dangerous materials, thanks to its precision control and stepper motor power.

Figure 3.14: Clamp Hand

3.2.5 Power Components

- **Buck Converter:** The buck converter is used to adjust the voltage to the desired level, ensuring that components receive the appropriate power supply for optimal performance.

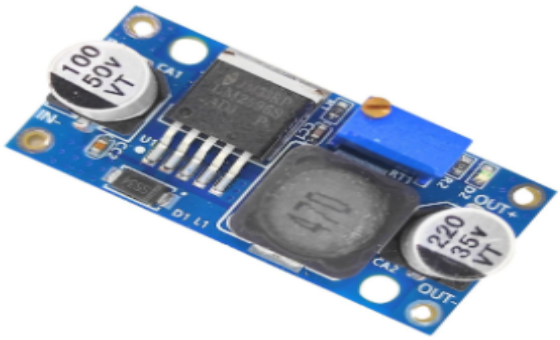


Figure 3.15: Buck Converter

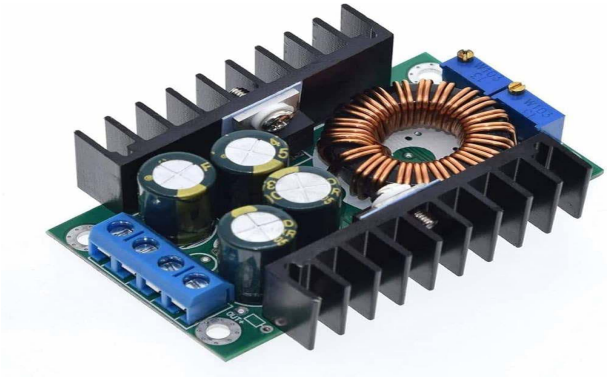


Figure 3.16: Another used Buck Converter Additional View

- **BMS 18650 lithium battery protection board 12V 20A:** The BMS (Battery Management System) regulates battery charging and discharging to ensure safe and efficient operation. It is used in this project to power the display, LEDs, robot's eyes, and cooling fan, providing stable energy distribution for these components.

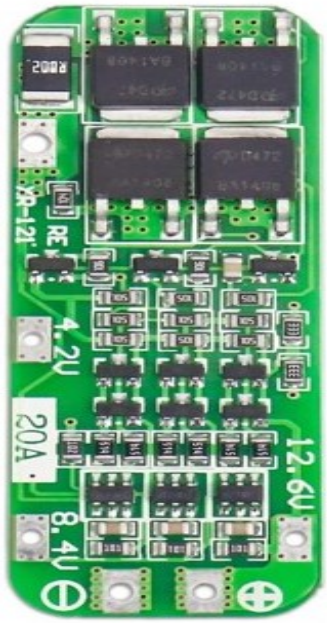


Figure 3.17: BMS 18650

- **5S BMS lithium battery protection board 16.8V 100A :** It is used to power the robot's wheels (legs), providing the necessary energy to handle heavy loads and ensure smooth and reliable movement.

Figure 3.18: 5S BMS

- **3S BMS 40A lithium battery protection 12V:** The 3S BMS regulates the charging and discharging of the lithium battery for the robot's arm. It ensures safe and efficient power delivery, enabling smooth and precise operation of the arm's movements.

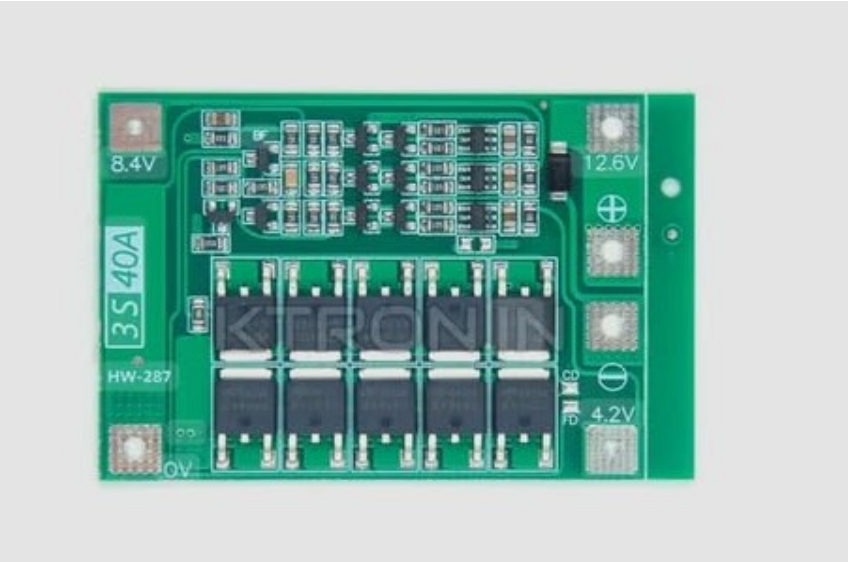


Figure 3.19: 3S BMS

3.2.6 Switches and Controls

- **Joysticks:**The joystick provides precise control over the robot's arm and head, allowing smooth movement in all directions. For the arm, it enables right, left, up, down, and 360-degree rotation, ensuring accurate positioning for gripping and manipulating objects. Similarly, for the head, the joystick controls 360-degree rotation and up-down movement, enhancing the robot's ability to scan its surroundings and adjust the camera angle for optimal visibility.

Figure 3.20: Joysticks

Figure 3.21: Joysticks2

- **Toggle Switch and push button:** is used extensively in the robot to control multiple relays, enabling and disabling various components efficiently. Each switch corresponds to a specific relay, allowing manual activation of critical functions such as motors and sensors.

Figure 3.22: Toggle Switch

Figure 3.23: push button Switch

- **emergency button :** an emergency button to press it in the Emergency Cases it stops all the robot .

Figure 3.24: emergency button

3.2.7 Communication Modules

- **NRF24:** is used for wireless communication, enabling remote control of the robot. It facilitates seamless transmission of commands to control the arm's movement and GPS data

Figure 3.25: NRF24

3.2.8 Displays

- **LCD Display for Eyes :** to display a friendley shape of eyes , that moves as a humen eyes

Figure 3.26: LCD Display for Eyes

- **screen Display:** This screen is placed in the robots belly , its displayed the information that we want to view about the robot (we put the information in a USB).

Figure 3.27: screen Display

- **1.8 TFT 128 * RGB * 160 ver 1V:** This screen is located in the remote control , it viewed the information received from the GPS.

Figure 3.28: 1.8 TFT

3.2.9 Auxiliary Components

- **Relay 4 Channel:** used to control multiple high-power components in the robot in motors. It allows switching different systems on and off remotely, ensuring efficient power management and seamless operation of critical functions.

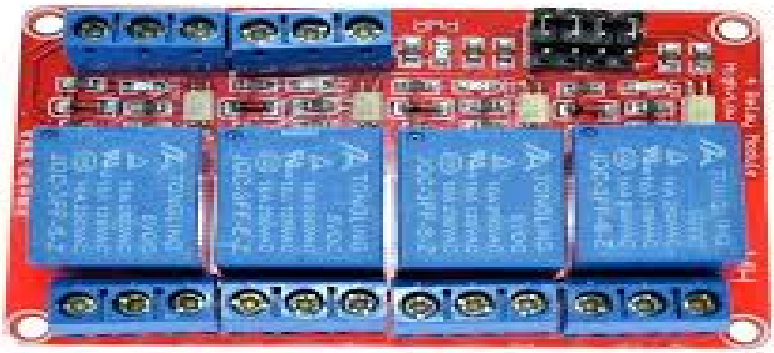


Figure 3.29: Relay 4 Channel

- **Fire Extinguisher:** The robot's main firefighting tool is the fire extinguisher. The drill motor engages to release the extinguishing agent when the sensor senses a fire. In dangerous situations, this guarantees quick and effective fire suppression.

Figure 3.30: Fire Extinguisher

- **Ball Bearing:** Ball bearings are used to reduce friction and ensure smooth rotational motion in the robot's moving parts, such as wheels and joints. They enhance durability and efficiency by allowing components to move freely and withstand heavy loads.

Figure 3.31: Ball Bearing

- **Copper Plate Board:** Electronic components are soldered and connected using the copper plate board as a base. It offers a strong, conductive surface that guarantees reliable and effective electrical connections throughout the robot's circuit.

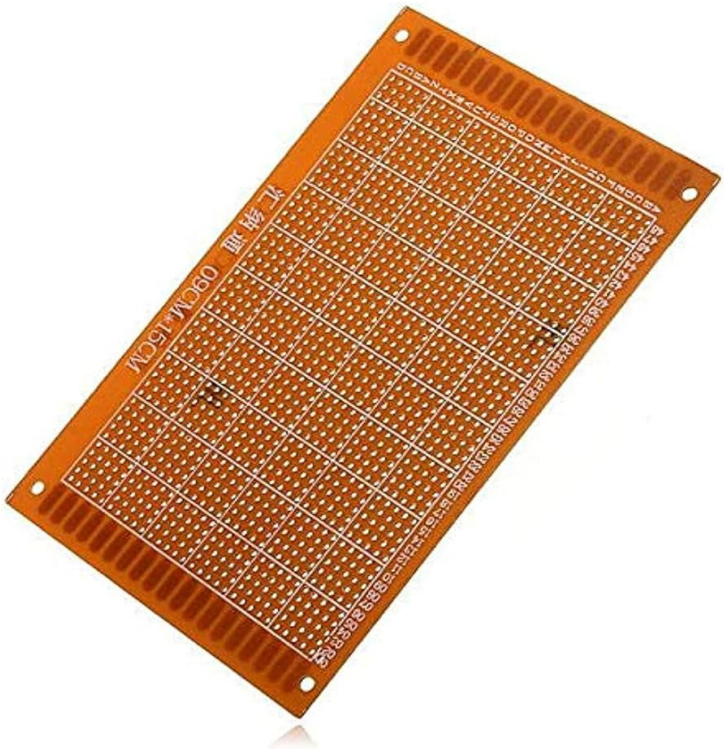


Figure 3.32: Copper Plate Board

- **Connectors:** Connectors are used to securely join wires and components, ensuring efficient and reliable electrical connections throughout the robot's system.

Figure 3.33: Connectors

- **Buzzer:** is used as a connection status indicator between the remote control and the robot. When the connection is stable, the buzzer remains silent. However, if the connection is lost, the buzzer activates, providing an audible alert to notify the operator of communication failure.

Figure 3.34: Buzzer

- **Cooling Fan:** A cooling fan for the system to ensure that the internal components, including the motors, batteries, and electronic circuits, are kept at a stable temperature, preventing overheating during extended operations.

Figure 3.35: Cooling Fan

- **Car Light:** this is used around the eyes on the head , to light the places around the robot especially when the vision is bad .

Figure 3.36: Car Light

3.2.10 Cameras

- **web cam:** This web cam is used for image Processing , it placed on the right hand where the clamp to detect the wire colors , To do the right thing for the expert .

Figure 3.37: web cam

- **FBV camera:** This camera is placed at the head of the robot , To give a live broadcast , And it rotates 360 degrees , it has its own network , we connect to this network in the mobile .

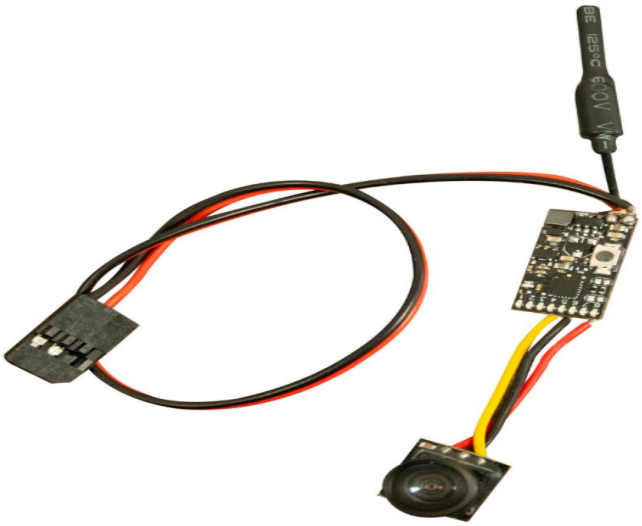


Figure 3.38: FBV camera

3.3 Circuit:

- **The mother Board :**

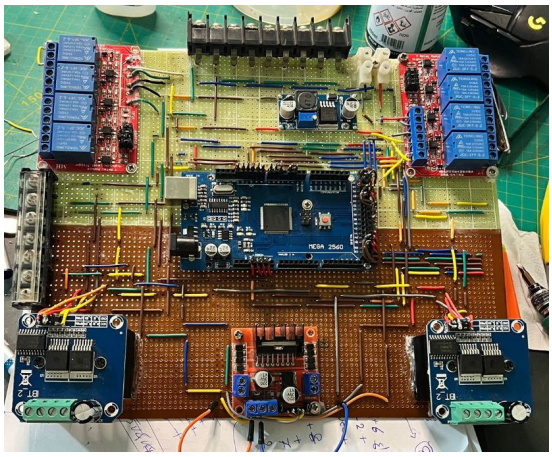


Figure 3.39: Mother Board front

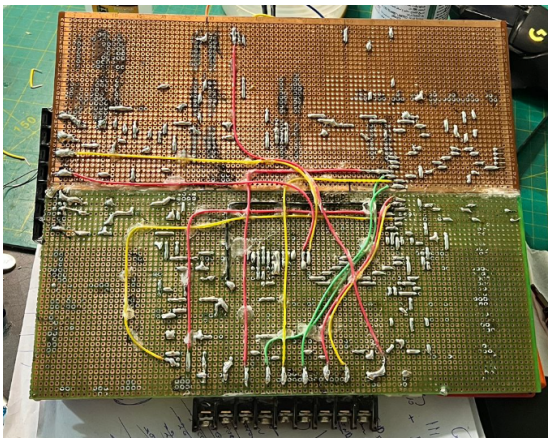


Figure 3.40: Mother Board back

- **The Remote Control Circuit :**

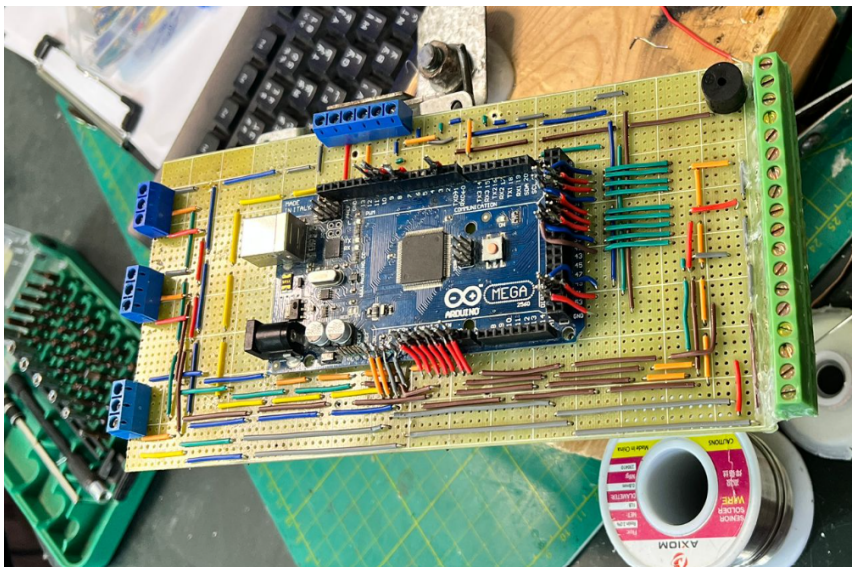


Figure 3.41: Remote Control Circuit

- **The Batteries we designed:**

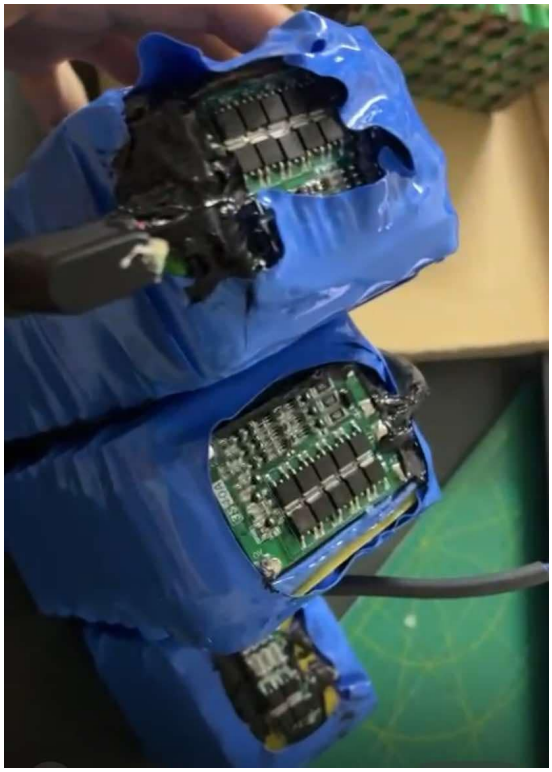


Figure 3.42: lithum batteries

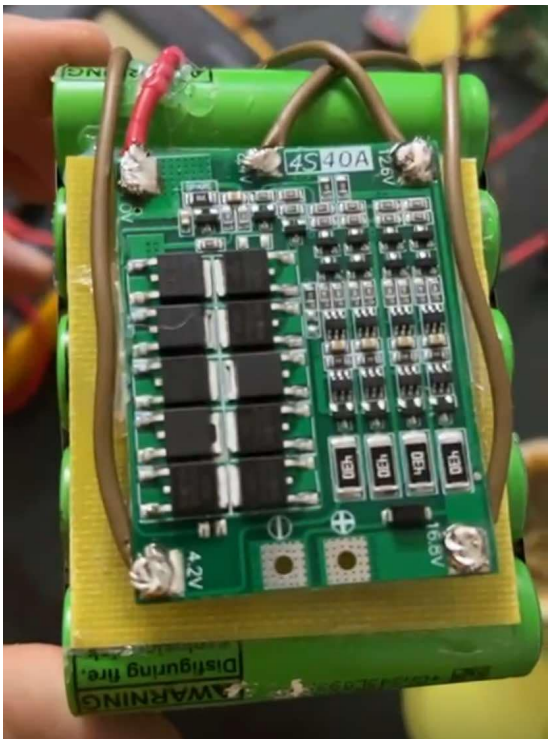


Figure 3.43: the Connections

3.4 Experimental Procedures

- **Right arm:**
  The right arm of the robot is designed for high versatility and precision. It can move in all directions—up, down, right, and left—and features an axis capable of 360-degree rotation, with an attached camera that rotates along with it to provide a complete field of view. At the end of the arm, two specialized claws are installed: one for cutting wires and the other for gripping or holding objects.

  The arm is powered by stepper motors configured for different gear ratios to ensure precise movement and control. The first motor operates with a 1:175 gear ratio, meaning 175 rotations of the motor produce one full rotation of the axis, while the second and third motors have gear ratios of 1:50 and 20:1, respectively, for enhanced torque and accuracy. The arm's operations are fully controlled remotely using a wireless remote control, allowing safe and efficient manipulation in hazardous environments.

- **Left arm:**
  this arm is fixed and non-movable like the right arm, has been innovatively repurposed into a fire suppression system. This system integrates two servo motors, each serving a critical role. The first servo motor is equipped with a fire sensor capable of detecting heat sources within a 180-degree range, continuously scanning for potential fire hazards.

  When a heat source is detected at a specific angle, the second servo motor, which is fixed, moves the fire extinguisher's nozzle toward the source. A signal is then sent to the driver of a drill motor connected to the fire extinguisher, which rotates to press and activate the extinguisher, releasing CO2 gas to suppress the fire. During this process, the fire sensor continues scanning for other heat sources, prioritizing the strongest one. Once the fire is extinguished, the system automatically signals the drill motor to stop and return the extinguisher to its original position.
  This fully automated fire suppression system ensures quick and effective responses to fire hazards, enhancing the robot's functionality in dangerous environments.

- **Legs:** The robot is equipped with two legs, right and left, which have been developed independently for enhanced flexibility and control. Each leg is operated using a separate joystick, allowing precise and individual control of each leg. These legs are specifically designed to handle the toughest conditions and navigate challenging terrains effectively.

  By separating the control of the legs, the robot ensures that if one leg becomes stuck, the other can independently assist in overcoming the obstacle. The legs are controlled remotely using a wireless remote control, providing seamless operation in difficult and hazardous environments.

- **Head:** It contains lighting to illuminate areas where vision is limited, and there are screens on it to give the appearance of grainy eyes. There is an FBV camera on it that provides a live broadcast to the phone, and it rotates 360 degrees to view the desired angle and also moves up and down.

- **Control System:** The control system is divided into two main components: the motherboard and the remote control, which work together to manage the robot's movements and operations.

  - **The motherboard :** The motherboard is the central hub that connects all the motors, Arduino Mega, drivers, and relays. These components are integrated in a way that allows seamless control of all parts of the robot through the remote control.

  - **The remote control :** The remote control features a custom control panel equipped with joysticks, switches, an emergency button, push buttons, and a screen. The screen provides real-time information about the robot, including its GPS location and status. This control panel enables precise and efficient management of the robot, allowing operators to navigate and control its movements and functions in hazardous environments.

- **Battery System:** The robot's power system is built with four specialized batteries, each designed to handle different components and operations effectively:

  - **Battery 1 (12V, 20A):** This battery consists of three cells, each containing eight 2.6A batteries connected in parallel to achieve the required amperage. The cells themselves are connected in series to provide the required voltage. This battery powers essential components such as the LEDs, GPS, servos, and eyes (components requiring power through the relay system).

  - **Battery 2 (12V, 12A):** This battery differs in type from the first. It is made up of six cells, each containing four 1.8A batteries connected in parallel to achieve the necessary voltage. The cells are then connected in series. This battery is responsible for powering the motherboard, drivers, Arduino Mega, and relays.

  - **Battery 3 (12V, 30A):** Designed specifically for the operation of the robot's arm, this battery comprises three cells, with each cell containing twelve batteries connected in parallel. The cells are connected in series to achieve the required output.

  - **Battery 4 (16.8V, 40A):**

    This battery is used to power the robot's legs. It consists of three cells, with each cell containing twenty batteries connected in parallel. The cells are connected in series to provide the necessary voltage and amperage.

    Each battery is equipped with a suitable Power Management System (PMS), except for the fourth battery. The PMS for the fourth battery is avoided because it powers motors that require a significantly high start current, consuming 3-5 times their normal operating current.

- **GPS System:** The GPS system in the robot provides comprehensive data about its movement and precise location. It outputs the coordinates in terms of latitude and longitude, which are displayed on the screen integrated into the remote control. These coordinates can be entered into any mapping service, such as Google Maps, to pinpoint the exact location of the robot on the map, ensuring accurate tracking and situational awareness during operation.

3.5 Codes

Control Code :

```
#include <SPI.h>
#include <RF24.h>

#define CE_PIN 9
#define CSN_PIN 10
#define EMERGENCY_BUTTON_PIN 7

RF24 radio(CE_PIN, CSN_PIN);

#define JOY_Y1 A0
#define JOY_Y2 A1
#define JOY_X3 A2
#define JOY_Y3 A3

#define BUTTON1 26
#define BUTTON2 27
#define BUTTON3 28
#define BUTTON4 29
#define BUTTON5 33
#define BUTTON6 35
#define BUTTON7 37
#define BUZZER_PIN 2
int joyY1, joyY2, joyX3, joyY3;

const uint64_t address = 0xF0F0F0F0E1LL;

bool connectionEstablished = false;

void setup() {

    pinMode(EMERGENCY_BUTTON_PIN, INPUT_PULLUP);

  pinMode(BUTTON1, INPUT_PULLUP);
  pinMode(BUTTON2, INPUT_PULLUP);
  pinMode(BUTTON3, INPUT_PULLUP);
  pinMode(BUTTON4, INPUT_PULLUP);
  pinMode(BUTTON5, INPUT_PULLUP);
  pinMode(BUTTON6, INPUT_PULLUP);
  pinMode(BUTTON7, INPUT_PULLUP);

  pinMode(BUZZER_PIN, OUTPUT);

  radio.begin();
  radio.setPALevel(RF24_PA_HIGH);
  radio.setChannel(0x4c);
  radio.openWritingPipe(address);
  radio.setDataRate(RF24_1MBPS);
  Serial.begin(9600);
  Serial.println("Transmitter ready...");
}

void loop() {
  joyY1 = analogRead(JOY_Y1) - 512;
  joyY2 = analogRead(JOY_Y2) - 512;
```

```
  joyX3 = analogRead(JOY_X3) - 512;
  joyY3 = analogRead(JOY_Y3) - 512;

  int button1 = digitalRead(BUTTON1);
  int button2 = digitalRead(BUTTON2);
  int button3 = digitalRead(BUTTON3);
  int button4 = digitalRead(BUTTON4);
  int button5 = digitalRead(BUTTON5);
  int button6 = digitalRead(BUTTON6);
  int button7 = digitalRead(BUTTON7);
  int emergencyButtonState = digitalRead(EMERGENCY_BUTTON_PIN);

  int data[12] = {joyY1, joyY2, joyX3, joyY3, button1, button2, button3,
     button4, button5, button6, button7, emergencyButtonState};

  bool success = radio.write(&data, sizeof(data));

  if (success) {
    if (!connectionEstablished) {
      connectionEstablished = true;

      tone(BUZZER_PIN, 2000);
      delay(2000);
      noTone(BUZZER_PIN);
    }
  } else {
    connectionEstablished = false;

    tone(BUZZER_PIN, 2000);
    delay(200);
    noTone(BUZZER_PIN);
    delay(200);
  }

  Serial.print("JoyY1: ");
  Serial.print(joyY1);
  Serial.print(" JoyY2: ");
  Serial.print(joyY2);
  Serial.print(" JoyX3: ");
  Serial.print(joyX3);
  Serial.print(" JoyY3: ");
  Serial.println(joyY3);

  delay(200);
}
```

Mother Boared code :

```
#include <SPI.h>
#include <RF24.h>
#include <Servo.h>

#define CE_PIN 9
#define CSN_PIN 10


RF24 radio(CE_PIN, CSN_PIN);

```

```
#define RELAY8 5
#define RELAY9 6
#define RELAY10 12
#define RELAY11 13
#define MOTOR3_PIN1 7
#define MOTOR3_PIN2 8


#define RELAY1 22
#define RELAY2 23
#define RELAY3 24
#define RELAY4 25
#define RELAY5 30
#define RELAY6 31
#define RELAY7 32

Servo myServo;

const uint64_t address = 0xF0F0F0F0E1LL;

void stopAll() {
  digitalWrite(RELAY8, LOW);
  digitalWrite(RELAY9, LOW);
  digitalWrite(RELAY10, LOW);
  digitalWrite(RELAY11, LOW);
  analogWrite(MOTOR3_PIN1, 0);
  analogWrite(MOTOR3_PIN2, 0);
  digitalWrite(RELAY1, LOW);
  digitalWrite(RELAY2, LOW);
  digitalWrite(RELAY3, LOW);
  digitalWrite(RELAY4, LOW);
  digitalWrite(RELAY5, LOW);
  digitalWrite(RELAY6, LOW);
  digitalWrite(RELAY7, LOW);
}


void setup() {
  pinMode(RELAY1, OUTPUT);
  pinMode(RELAY2, OUTPUT);
  pinMode(RELAY3, OUTPUT);
  pinMode(RELAY4, OUTPUT);
  pinMode(RELAY5, OUTPUT);
  pinMode(RELAY6, OUTPUT);
  pinMode(RELAY7, OUTPUT);

  pinMode(RELAY8, OUTPUT);
  pinMode(RELAY9, OUTPUT);
  pinMode(RELAY10, OUTPUT);
  pinMode(RELAY11, OUTPUT);

  pinMode(MOTOR3_PIN1, OUTPUT);
  pinMode(MOTOR3_PIN2, OUTPUT);

  myServo.attach(11);
  myServo.write(30);

  radio.begin();
```

```
  radio.setPALevel(RF24_PA_HIGH);
  radio.setChannel(0x4c);
  radio.openReadingPipe(1, address);
  radio.setDataRate(RF24_1MBPS);
  radio.startListening();

  Serial.begin(9600);
  Serial.println("Receiver ready...");
}

void loop() {
  if (radio.available()) {
    int data[12];
    radio.read(&data, sizeof(data));

    int joyY1 = data[0];
    int joyY2 = data[1];
    int joyX3 = data[2];
    int joyY3 = data[3];
    int button1 = data[4];
    int button2 = data[5];
    int button3 = data[6];
    int button4 = data[7];
    int button5 = data[8];
    int button6 = data[9];
    int button7 = data[10];
    int emergencyButtonState = data[11];

    if (emergencyButtonState == HIGH) {
      stopAll();
      return;
    }



if (joyY1 > 100) {
    digitalWrite(RELAY8, HIGH);
    digitalWrite(RELAY9, LOW);
} else if (joyY1 < -100) {
    digitalWrite(RELAY9, HIGH);
    digitalWrite(RELAY8, LOW);
} else {
    digitalWrite(RELAY8, LOW);
    digitalWrite(RELAY9, LOW);
}

if (joyY2 > 100) {
    digitalWrite(RELAY10, HIGH);
    digitalWrite(RELAY11, LOW);
} else if (joyY2 < -100) {
    digitalWrite(RELAY11, HIGH);
    digitalWrite(RELAY10, LOW);
} else {
    digitalWrite(RELAY10, LOW);
    digitalWrite(RELAY11, LOW);
```

```
    if (joyX3 > 200) {
      analogWrite(MOTOR3_PIN1, map(joyX3, 200, 512, 0, 255));
      analogWrite(MOTOR3_PIN2, 0);
    } else if (joyX3 < -200) {
      analogWrite(MOTOR3_PIN2, map(abs(joyX3), 200, 512, 0, 255));
      analogWrite(MOTOR3_PIN1, 0);
    } else {
      analogWrite(MOTOR3_PIN1, 0);
      analogWrite(MOTOR3_PIN2, 0);
    }

    int servoAngle = map(joyY3, -512, 512, 0, 180);
    myServo.write(constrain(servoAngle, 0, 180));

    digitalWrite(RELAY1, button1 == LOW ? LOW : HIGH);
    digitalWrite(RELAY2, button2 == LOW ? LOW : HIGH);
    digitalWrite(RELAY3, button3 == LOW ? LOW : HIGH);
    digitalWrite(RELAY4, button4 == LOW ? LOW : HIGH);
    digitalWrite(RELAY5, button5 == LOW ? LOW : HIGH);
    digitalWrite(RELAY6, button6 == LOW ? HIGH : LOW);

    digitalWrite(RELAY7, button7 == LOW ? LOW : HIGH);
  }
}
```

for GPS Sender :

```
TinyGPSPlus gps;
SoftwareSerial gpsSerial(4, 3);

#define CE_PIN 9
#define CSN_PIN 10
RF24 radio(CE_PIN, CSN_PIN);
const byte address[6] = "00003";

struct GPSData {
  float lat;
  float lng;
  float speed;
  float altitude;
};

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

  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_LOW);
  radio.setChannel(0x4e);
  radio.setDataRate(RF24_250KBPS);
  radio.stopListening();
  Serial.println("GPS Sender Initialized");
}

void loop() {
  while (gpsSerial.available()) {
    char c = gpsSerial.read();
```

```
    gps.encode(c);
  }

  if (gps.location.isUpdated()) {
    GPSData data;
    data.lat = gps.location.lat();
    data.lng = gps.location.lng();
    data.speed = gps.speed.kmph();
    data.altitude = gps.altitude.meters();

    bool success = radio.write(&data, sizeof(data));
    if (success) {
      Serial.println("Data Sent Successfully!");
    } else {
      Serial.println("Failed to Send Data");
    }

    delay(1000);
  }
}
```

for GPS Resever :

```

#define TFT_CS 10
#define TFT_RST 9
#define TFT_DC 8
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

#define CE_PIN 6
#define CSN_PIN 7
RF24 radio(CE_PIN, CSN_PIN);
const byte address[6] = "00003";

struct GPSData {
  float lat;
  float lng;
  float speed;
  float altitude;
};

#define BACKGROUND ST7735_BLACK
#define TEXT_COLOR ST7735_WHITE
#define VALUE_COLOR ST7735_YELLOW
#define WORD1_COLOR ST7735_WHITE
#define WORD2_COLOR ST7735_BLUE
#define WORD3_COLOR ST7735_RED
#define WORD4_COLOR ST7735_YELLOW

bool dataAvailable = false;

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

  tft.initR(INITR_BLACKTAB);
  tft.fillScreen(BACKGROUND);
  tft.setRotation(1);
  tft.setTextColor(TEXT_COLOR);
```

```
  tft.setTextSize(1);
  tft.setCursor(10, 30);
  tft.println("Initializing...");
  delay(3000);

  tft.fillScreen(BACKGROUND);

  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_LOW);
  radio.setChannel(0x4e);
  radio.setDataRate(RF24_250KBPS);
  radio.startListening();

  Serial.println("GPS Receiver Initialized");
}

void displayWaitingMessage() {
  tft.fillScreen(BACKGROUND);

  tft.setTextColor(WORD1_COLOR);
  tft.setTextSize(1);
  tft.setCursor(10, 30);
  tft.print("Waiting");

  tft.setTextColor(WORD2_COLOR);
  tft.setCursor(60, 30);
  tft.print("for");
  tft.setTextColor(WORD3_COLOR);
  tft.setCursor(90, 30);
  tft.print("GPS");

  tft.setTextColor(WORD4_COLOR);
  tft.setCursor(120, 30);
  tft.print("Data");

tft.setTextColor(VALUE_COLOR);
  tft.setTextSize(1.4);
  tft.setCursor(10, 80);
  tft.println("  WALL-E Savior-ROBOT");

}

void displayGPSData(GPSData data) {
  tft.fillScreen(BACKGROUND);

  tft.setTextColor(TEXT_COLOR);
  tft.setTextSize(1.5);
  tft.setCursor(5, 10);
  tft.println("GPS Data:");

  tft.setTextColor(VALUE_COLOR);
  tft.setCursor(5, 30);
  tft.print("Lat:");
  tft.println(data.lat, 6);

  tft.setTextColor(WORD2_COLOR);
  tft.setCursor(5, 50);
```

```
  tft.print("Lng:");
  tft.println(data.lng, 6);

  tft.setTextColor(WORD3_COLOR);
  tft.setCursor(90, 30);
  tft.print("Speed:");
  tft.println(data.speed, 2);

  tft.setTextColor(TEXT_COLOR);
  tft.setCursor(90,50);
  tft.print("Alt:");
  tft.println(data.altitude, 2);

  tft.setTextColor(VALUE_COLOR);
  tft.setTextSize(2);
  tft.setCursor(10, 80);
  tft.println("WALL-E ROBOT");
}

void loop() {
  if (radio.available()) {
    GPSData data;
    radio.read(&data, sizeof(data));
    Serial.println("Data Received!");

    dataAvailable = true;

    displayGPSData(data);

    Serial.print("Lat: ");
    Serial.println(data.lat, 6);
    Serial.print("Lng: ");
    Serial.println(data.lng, 6);
    Serial.print("Speed: ");
    Serial.println(data.speed, 2);
    Serial.print("Alt: ");
    Serial.println(data.altitude, 2);
  } else {
    if (!dataAvailable) {
      displayWaitingMessage();
    }
  }

  delay(500);
}
```

Sender Arm

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#define CE_PIN 9
#define CSN_PIN 10

const int potXPin = A0;
const int potYPin = A1;
const int potZPin = A2;
```

```
const int btnJoystickPin = 2;
const int btnExternalPin = 3;

const int ledPin = 7;

RF24 radio(CE_PIN, CSN_PIN);

const byte address[6] = "00002";

struct DataPacket {
  int potX;
  int potY;
  int potZ;
  bool joyBtnPressed;
  bool extBtnPressed;
};

bool isConnected = false;

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

  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.setChannel(0x4d);
  radio.setDataRate(RF24_250KBPS);
  radio.stopListening();

  pinMode(btnJoystickPin, INPUT_PULLUP);
  pinMode(btnExternalPin, INPUT_PULLUP);

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}

void loop() {
  DataPacket data;
  data.potX = analogRead(potXPin);
  data.potY = analogRead(potYPin);
  data.potZ = analogRead(potZPin);
  data.joyBtnPressed = (digitalRead(btnJoystickPin) == LOW);
  data.extBtnPressed = (digitalRead(btnExternalPin) == LOW);

  bool isDataSent = radio.write(&data, sizeof(data));

  if (isDataSent) {
    if (!isConnected) {
      isConnected = true;
      digitalWrite(ledPin, HIGH);
    }
  } else {
    if (isConnected) {
      isConnected = false;
      digitalWrite(ledPin, LOW);
    }
    digitalWrite(ledPin, HIGH);
    delay(100);
```

```
    digitalWrite(ledPin, LOW);
    delay(100);
  }

  delay(1);
}
```

Receiver Arm

```
#define CE_PIN 9
#define CSN_PIN 10

const int stepXPin = 2;
const int dirXPin  = 3;
const int enaXPin  = 4;

const int stepYPin = 5;
const int dirYPin  = 6;
const int enaYPin  = 7;

const int stepZPin = A0;
const int dirZPin  = A1;
const int enaZPin  = A2;

const int servo1Pin = A3;
const int servo2Pin = A4;

const int JOY_CENTER_X = 512;
const int JOY_CENTER_Y = 512;
const int JOY_CENTER_Z = 470;
const int DEADZONE     = 100;

const float gearX = 175.0;
const float gearY = 50.0;
const float gearZ = 19.0;

const float rangeXmin = 0.0,   rangeXmax = 90.0;
const float rangeYmin = 0.0,   rangeYmax = 180.0;
const float rangeZmin = 0.0,   rangeZmax = 360.0;

const float startAngleX = 90.0;
const float startAngleY = 180.0;
const float startAngleZ = 0.0;

const float motorStepsPerRev = 1600.0;

const float accelAll     = 5000.0;
const float fixedSpeedX  = 25000.0;
const float fixedSpeedY  = 20000.0;
const float fixedSpeedZ  = 15000.0;

const int servo1HomeAngle  = 0;
const int servo1PressAngle = 90;
const int servo2HomeAngle  = 0;
const int servo2PressAngle = 90;

AccelStepper stepperX(AccelStepper::DRIVER, stepXPin, dirXPin);
```

```
AccelStepper stepperY(AccelStepper::DRIVER, stepYPin, dirYPin);
AccelStepper stepperZ(AccelStepper::DRIVER, stepZPin, dirZPin);
Servo servo1, servo2;

RF24 radio(CE_PIN, CSN_PIN);

const byte address[6] = "00002";

struct DataPacket {
  int potX;
  int potY;
  int potZ;
  bool joyBtnPressed;
  bool extBtnPressed;
};

int filteredPotZ = JOY_CENTER_Z;

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

  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.setChannel(0x4d);
  radio.setDataRate(RF24_250KBPS);
  radio.startListening();

  pinMode(enaXPin, OUTPUT); digitalWrite(enaXPin, LOW);
  pinMode(enaYPin, OUTPUT); digitalWrite(enaYPin, LOW);
  pinMode(enaZPin, OUTPUT); digitalWrite(enaZPin, LOW);

  stepperX.setMaxSpeed(fixedSpeedX);
  stepperX.setAcceleration(accelAll);
  stepperY.setMaxSpeed(fixedSpeedY);
  stepperY.setAcceleration(accelAll);
  stepperZ.setMaxSpeed(fixedSpeedZ);
  stepperZ.setAcceleration(accelAll);

  stepperX.setCurrentPosition((long)(startAngleX * (motorStepsPerRev *
     gearX) / 360.0));
  stepperY.setCurrentPosition((long)(startAngleY * (motorStepsPerRev *
     gearY) / 360.0));
  stepperZ.setCurrentPosition((long)(startAngleZ * (motorStepsPerRev *
     gearZ) / 360.0));

  servo1.attach(servo1Pin);
  servo2.attach(servo2Pin);
  servo1.write(servo1HomeAngle);
  servo2.write(servo2HomeAngle);

}

void loop() {
  if (radio.available()) {
    DataPacket data;
    radio.read(&data, sizeof(data));

```

```
    filteredPotZ = 0.8 * filteredPotZ + 0.2 * data.potZ;
    data.potZ = filteredPotZ;

    int potX = data.potX - JOY_CENTER_X;
    int potY = data.potY - JOY_CENTER_Y;
    int potZ = data.potZ - JOY_CENTER_Z;

    float spdX = 0, spdY = 0, spdZ = 0;

    if (potX >  DEADZONE)      spdX = +fixedSpeedX;
    else if (potX < -DEADZONE) spdX = -fixedSpeedX;
    else                       spdX = 0;

    if (potY >  DEADZONE)      spdY = +fixedSpeedY;
    else if (potY < -DEADZONE) spdY = -fixedSpeedY;
    else                       spdY = 0;

    if (potZ >  DEADZONE)      spdZ = +fixedSpeedZ;
    else if (potZ < -DEADZONE) spdZ = -fixedSpeedZ;
    else                       spdZ = 0;

    stepperX.setSpeed(spdX);
    stepperY.setSpeed(spdY);
    stepperZ.setSpeed(spdZ);

    stepperX.run();
    stepperY.run();
    stepperZ.run();

    if (data.joyBtnPressed)  servo1.write(servo1PressAngle);
    else                     servo1.write(servo1HomeAngle);

    if (data.extBtnPressed)  servo2.write(servo2PressAngle);
    else                     servo2.write(servo2HomeAngle);
  }

  delay(1);
}
```

Fire system :

```
const int flameSensorPin = A0;
const int flameThreshold = 300;
const int servo2Pin = 10;
const int IN1 = 7;
const int IN2 = 6;

Servo myServo;
Servo servo2;

bool isFireDetected = false;
unsigned long lastMotorTime = 0;
bool motorRunning = false;
int motorDirection = 0;

void setup() {
  pinMode(flameSensorPin, INPUT);
  pinMode(IN1, OUTPUT);
```

```
  pinMode(IN2, OUTPUT);

  Serial.begin(9600);
  myServo.attach(11);
  servo2.attach(servo2Pin);

  stopMotor();
}

void loop() {
  static int angle = 20;
  static bool increasing = true;
  static unsigned long lastServoMoveTime = 0;

  if (millis() - lastServoMoveTime >= 30) {
    lastServoMoveTime = millis();
    myServo.write(angle);

    if (increasing) {
      angle++;
      if (angle >= 160) increasing = false;
    } else {
      angle--;
      if (angle <= 20) increasing = true;
    }
  }

  int flameValue = analogRead(flameSensorPin);
  Serial.print("          : ");
  Serial.print(angle);
  Serial.print(", Flame Value: ");
  Serial.println(flameValue);

  if (flameValue < flameThreshold) {
    if (!isFireDetected) {
      Serial.println("Fire detected! Moving forward...");
      startMotor(1, 3000);
      isFireDetected = true;
    }
    servo2.write(angle);
  } else {
    if (isFireDetected) {
      Serial.println("No fire detected! Moving backward...");
      startMotor(-1, 3000);
      isFireDetected = false;
    }
  }

  if (motorRunning && millis() - lastMotorTime >= 3000) {
    stopMotor();
    motorRunning = false;
  }
}

void startMotor(int direction, int duration) {
  motorRunning = true;
  motorDirection = direction;
  lastMotorTime = millis();
```

```

  if (direction == 1) {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
  } else if (direction == -1) {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
  }
}

void stopMotor() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
}
```

Wire color detection :

```
import cv2
import numpy as np

def detect_wire_colors(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Red color range
    red_lower = np.array([0, 120, 70])
    red_upper = np.array([10, 255, 255])
    mask_red = cv2.inRange(hsv, red_lower, red_upper)

    # Blue color range
    blue_lower = np.array([100, 150, 70])
    blue_upper = np.array([140, 255, 255])
    mask_blue = cv2.inRange(hsv, blue_lower, blue_upper)

    # Green color range
    green_lower = np.array([40, 70, 70])
    green_upper = np.array([80, 255, 255])
    mask_green = cv2.inRange(hsv, green_lower, green_upper)

    min_length = 50  # Adjust this value based on calibration for 5 cm

    # Red wire detection
    contours_red, _ = cv2.findContours(mask_red, cv2.RETR_TREE, cv2.
       CHAIN_APPROX_SIMPLE)
    for contour in contours_red:
        area = cv2.contourArea(contour)
        if area > 100:
            x, y, w, h = cv2.boundingRect(contour)
            if max(w, h) >= min_length:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255),
                   2)
                cv2.putText(frame, "Red Wire", (x, y - 10), cv2.
                   FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    # Blue wire detection
    contours_blue, _ = cv2.findContours(mask_blue, cv2.RETR_TREE, cv2.
       CHAIN_APPROX_SIMPLE)
    for contour in contours_blue:
        area = cv2.contourArea(contour)
```

```
        if area > 100:
            x, y, w, h = cv2.boundingRect(contour)
            if max(w, h) >= min_length:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0),
                    2)
                cv2.putText(frame, "Blue Wire", (x, y - 10), cv2.
                    FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)

    # Green wire detection
    contours_green, _ = cv2.findContours(mask_green, cv2.RETR_TREE, cv2.
        CHAIN_APPROX_SIMPLE)
    for contour in contours_green:
        area = cv2.contourArea(contour)
        if area > 100:
            x, y, w, h = cv2.boundingRect(contour)
            if max(w, h) >= min_length:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0),
                    2)
                cv2.putText(frame, "Green Wire", (x, y - 10), cv2.
                    FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

    return frame

def main():
    print("Wire Color Detection")

    # Start video capture (use 0 for default camera)
    cap = cv2.VideoCapture(0, cv2.CAP_V4L2)  # Use V4L2 backend for
        Raspberry Pi

    if not cap.isOpened():
        print("Error: Unable to access the camera.")
        return

    print("Press 'q' to exit.")
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: Unable to capture frame.")
            break

        output = detect_wire_colors(frame)

        # Display the output
        cv2.imshow("Wire Color Detection", output)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if _name_ == "_main_":
    main()
```

Chapter 4

Discussion

4.1 Mechanical Challenges

- **Unavailability and High Cost of Components:** One of the biggest challenges that we faced is the unavailability and high cost of certain mechanical components. For example, one of the robot's legs differs from the other one . on the first leg we were able to find a chain with the correct spacing between its links, suitable for our design. This allowed us to use bicycle gears to reduce costs. We adjusted the number of teeth on the gears to match the available chain.

  However, for the second leg, we couldn't find an identical chain as the first leg. Instead we used a different chain and tried to simulate the functionality of the first one. To solve the mismatch, we added a support wheels to reduce mechanical variations. although these efforts, the second chain sometimes slips off the gears , because it does not perfectly fit. This issue is further being worse by the gears being relatively thin and prone to slight bending, which affects the chain's alignment and smooth operation on its path. While this problem does not always occur, it remains a mechanical challenge we faced.

- **Insufficient Motor Power:** Another big challenge we faced was the lack of sufficiently powerful motors to move the robot effectively. Initially, we tested several motors but found that most dosen't have the required strength. As a cost-effective solution, we choose to use a car wiper motors, as they are both powerful and affordable compared to other alternatives.

  However, even the wiper motors presented some limitations.The internal gears of these motors are made of plastic, which can withstand only a certain level of stress before they begin to degrade or fail. We experienced several issues related to gear durability during testing. After extensive trials with different motors, we find a suitable wiper motor that met our requirements, trying to find a balance between affordability and functionality. Although the limitations of the internal plastic gears remain a consideration for long-term usage and durability.

- **ball bearing problem** The problem with the ball bearing is that it cannot withstand welding or high temperatures. We had a specific design in mind, but were forced to change it and use bicycle gears instead, as they contain ball bearings that are capable of withstand heat and welding.

4.2 Electrical Challenges

- **Battery Specifications:** One of the main challenges we faced was the unavailability of batteries that met the requirements for our project. We needed batteries with high voltage capacity and lightweight construction, but such batteries were both expensive and unavailable in our country. Initially, we considered using solar panel batteries; however, their heavy weight made them unsuitable for our project.

  As a result, we had to design and assemble the batteries ourselves. Instead of creating a single battery, we designed multiple batteries to distribute the load, particularly the start current, across different systems in a balanced manner. This approach allowed us to optimize performance while working around the limitations of locally available resources.

- **Relay Noise Interference:** Another issue was the noise generated by the relays during operation, which significantly interfered with the wireless control system. Initially, we tried to solve this problem by adding diodes to mitigate noise; however, this solution was not effective. Ultimately, we had to separate the microcontroller to isolate interference and ensure smooth operation of the wireless control system. This workaround helped stabilize the system, although it added complexity to the overall design.

- **Insufficient Current Handling Capacity of the Motor Drivers:** The problem is that the DC motor drivers used are not capable of handling the high current over extended periods. although we select the best available drivers here , they struggle to maintain prolonged operation due to the high power demands required by the motors. The issue arises because drivers cannot efficiently tolerate the current supplied from the battery to the motor, leading to potential overheating and performance

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The WALL-E Savior Robot project demonstrates the potential of robotics and advanced technologies in addressing critical safety and operational challenges in hazardous environments. By integrating fire detection and suppression systems, a precision manipulator for wire handling, and wireless remote control capabilities, the robot is designed to reduce human exposure to danger and enhance efficiency in high-risk scenarios. The use of rechargeable batteries ensures extended operational reliability, while the inclusion of image processing technologies enables accurate and effective task execution.

This project highlights the transformative role of robotics in improving safety standards and operational effectiveness across industries. Whether in fire-fighting, bomb disposal, or other dangerous applications, the WALL-E Savior Robot represents a significant step toward leveraging technology for safer and more efficient solutions. It is a testament to the importance of innovation in creating tools that safeguard human lives while pushing the boundaries of what robotics can achieve in real-world applications.

5.2 What We Have Learned:

- **Integration of Components:** : We learned how to effectively integrate various hardware components, such as stepper motors, servos, and microcontrollers, to work together harmoniously in a complex system.

- **Power Management: :** Proper power management is crucial. We understood the importance of using separate power supplies for different components to ensure stable operation and prevent overload.

- **Importance of Precision:** Precise control over mechanical movements is vital. Using servo motors and stepper motors taught us the significance of accuracy in these applications especially that concerned with human safety.

- **Software Optimization:** Optimizing code for efficiency and reducing power consumption were crucial for extending the robot's operational time and enhancing overall performance.

- **Testing and Validation:** Testing the robot in various conditions underscored the importance of validating the system's performance across different scenarios to ensure reliability and robustness.

- **Maintenance and Troubleshooting:** Regular maintenance and troubleshooting skills were essential for keeping the robot operational and addressing any issues that arose during testing and deployment.

5.3 Future Work

- **Thermal Camera Integration:** Enables detection of humans to prioritize safety and assist in rescue operations.

- **Integration of LiDAR Sensor:** Adding a LiDAR sensor to enhance the robot's ability to map its surroundings in 3D, detect obstacles, and navigate complex environments with precision.

- **Integration of a 360-Degree Camera:** Adding a 360-degree camera would enable the robot to capture a complete view of its surroundings in real-time, eliminating the need for a servo to rotate. This would improve situational awareness, speed up decision-making, and enhance efficiency in dynamic and hazardous environments, such as disaster zones or rescue operations.

- **Microphone for Real-Time Communication:** Facilitates interaction between operators and individuals in hazardous zones, improving emergency coordination.

- **Enhanced Camera Functionality:** to enhance the Object detection (detect more useful objects that are dangerous).

- **Modular Design with Interchangeable Components:** Adaptable to various tasks, Increases utility for emergency responders(for example specialized arm tools to help dealing with dangerous objects).

- **AI and Machine Learning Integration:** Enhances data analysis, pattern recognition, and real-time decision-making. Reduces reliance on constant human supervision. Improves operational efficiency in critical scenarios.

- **Enhanced Fire Suppression System:** this includes developing a fire-extinguishing ball launcher on the robot's arm. This system will allow the arm to accurately throw fire-extinguishing balls at fire sources, improving efficiency and expanding the robot's ability to combat fires in the hazardous areas.

Bibliography

[1] Sung Hoon Kim, Sung Yon Park, Kae Won Choi, Tae-Jin Lee, and Dong In Kim. Backscatter-aided cooperative transmission in wireless-powered heterogeneous networks. *IEEE Transactions on Wireless Communications*, 19(11):7309–7323, 2020.

[2] Yong-tao Liu, Rui-zhi Sun, Tian-yi Zhang, Xiang-nan Zhang, Li Li, and Guo-qing Shi. Warehouse-oriented optimal path planning for autonomous mobile fire-fighting robots. *Security and Communication Networks*, 2020(1):6371814, 2020.