Quadruped project

By team members:

Arwa Sallam Mohamed 20200067

Zahraa Ahmed Abdelkafy 20201082

Esraa haitham ghareeb 20200077

Reem Ahmed Khalil 20201075

Riyad Abdelmoniem Attia 20190212

Mohamed Mashhoot Mohamed 20190478

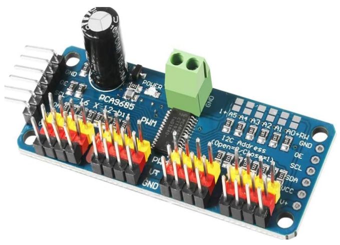
Abstract

The field of robotics has witnessed remarkable advancements in recent years, revolutionizing various aspects of our lives. This field has experienced rapid growth, enabling robots to perform complex tasks in dynamic and challenging environments. Hexapod robots, with their versatile locomotion capabilities, have played a significant role in this progression. They have emerged as a popular choice due to their stability, adaptability, and enhanced maneuverability. This report presents the design and implementation of a 4-legged spider robot equipped with servo-controlled legs. The proposed robot aims to leverage the benefits of hexapods in a compact and efficient form, offering potential applications in areas such as exploration, surveillance, and search-and-rescue operations. This report provides an overview of the background, the proposed design, the software architecture, and the future prospects of the spider robot.

Background

The primary objective of this project is to design and implement a 4-legged spider robot capable of performing various locomotion tasks using servo-controlled legs. The robot's mechanical structure consists of four legs, each equipped with two servos to control the leg joints. This configuration allows for precise leg movements, enabling the robot to walk, crawl, and potentially execute more complex maneuvers. The spider robot's compact design and advanced leg control mechanisms make it an ideal candidate for applications that require agility, such as exploration in confined spaces or surveillance in challenging environments.
Hardware components we used:

Servo Motor S3003: is a commonly used component in robotic applications. It is a standard-sized servo motor known for its affordability, durability, and ease of use. The S3003 provides rotational motion with a range of approximately 180 degrees. It offers precise control over leg movements, making it suitable for the leg joints of the spider robot.



PCA9685 Servo 16 Channel Driver: is a popular choice for controlling multiple servo motors simultaneously. It provides independent control over each servo motor connected to its 16 output channels. The servo driver communicates with the microcontroller using I2C protocol, simplifying the control interface.

ESP32: is a versatile microcontroller board based on the ESP32 system-on-a-chip. It combines powerful processing capabilities with built-in Wi-Fi and Bluetooth connectivity, making it suitable for controlling the spider robot and enabling wireless communication. It provides ample processing power to handle complex control algorithms and sensor integration. It can be programmed using the Arduino IDE.

Power Adapter: is used to provide the necessary power supply to the spider robot's electronic components, such as the servo motors, and servo driver. A dedicated power adapter ensures a stable and regulated power supply to the robot's components.

Disadvantages that happened:

- Dependency on External Power: The robot needs to be connected to the power adapter, limiting its mobility and requiring an available power source.
- Complexity: The PCA9685 servo driver adds an additional layer of complexity to the system, requiring proper configuration and software implementation.
- Limited Rotation: The S3003 servo motor has a limited rotation range of around 180 degrees, which may restrict the robot's range of motion.
- Lower Torque: The S3003 servo motor provides lower torque compared to larger servo motors, which may limit its ability to handle heavy loads or challenging terrains.

Modifications that we are likely to do:

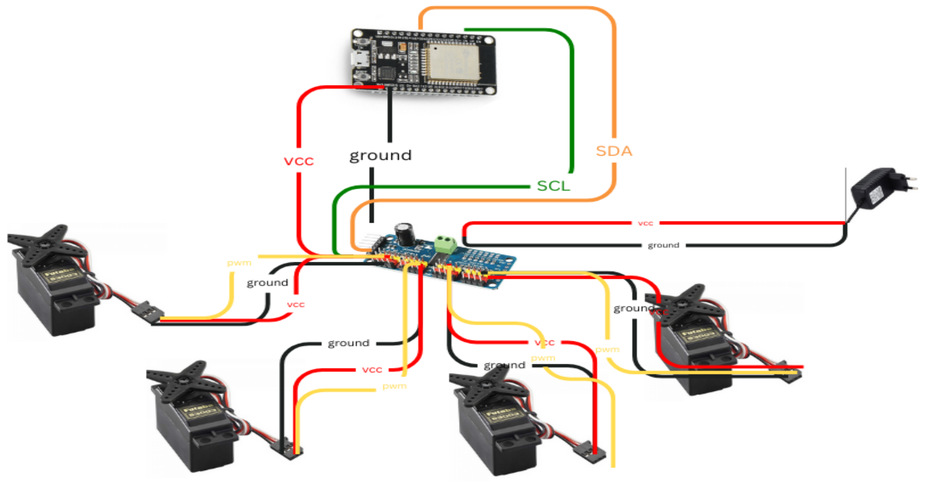
- An alternative to the S3003 servo motor could be higher torque servo motors such as the HS-755HB or the MG996R. These servo motors offer increased torque and are better suited for applications that require the robot to traverse rough terrains or carry heavier payloads.
- An alternative to a regular power adapter could be using a battery pack or rechargeable batteries. This alternative provides greater mobility and eliminates the need for a constant external power source.

Body Material:

the 3D frame is not fully implemented by us it’s modified from an already made 3D open source models on the GrabCAD website

Proposed idea

The development of a 4-legged spider robot with servo-controlled legs represents a significant step towards achieving versatile and efficient locomotion in robotics. The project demonstrates the feasibility of mimicking biological systems to enhance robotic mobility and adaptability.



In this picture we try to illustrate the Wiring diagram of the circuit but this is just for 4 motors

1. Electrical specifications of any electrical components:
    - 3D printed body of the spider
    - Esp32 div module
    - pca9685 servo 16 channel
    - 8 s3003 servo motors
    - Power adapter 5v 2A

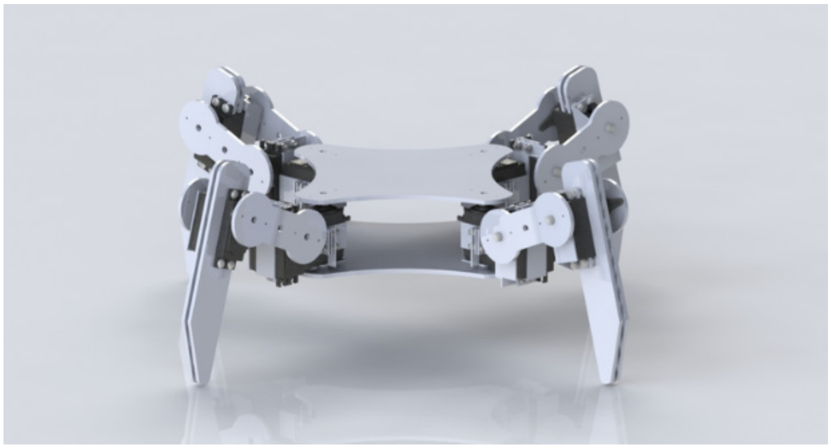
2. The design of the body:

- 2 base
- 8 gripper_2
- 4 gripper_4
- 4 gripper_6
- 8 leg
- 4 thi

Here are the stl files of the design :
https://drive.google.com/drive/folders/1V3CuaXYz4ktOqubByVTiCxFekPJBMCQ9?usp=sharing

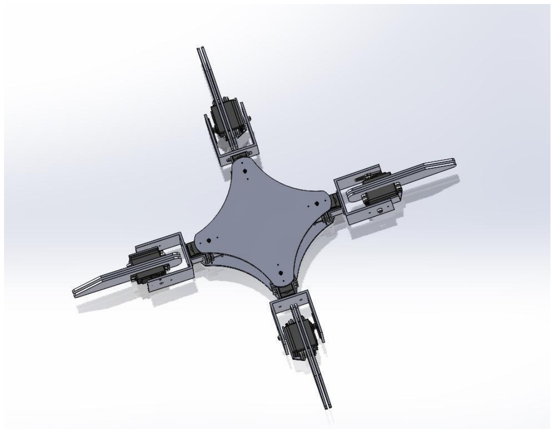
This is the design we chose from GrabCAD website:

But this design has 3 servos in each leg, and we have decided to use 2 servos in the leg, so we made some modifications on the design.

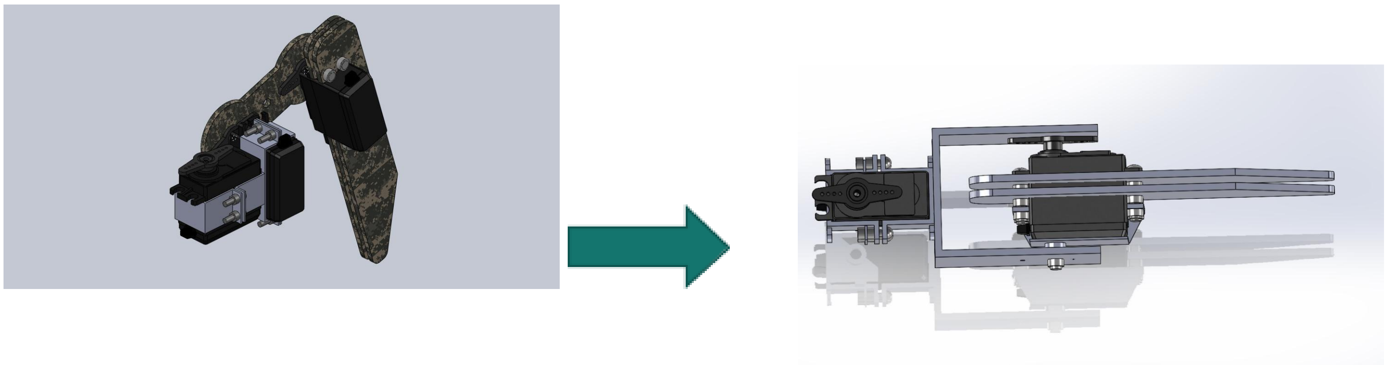


This is our robot design after editing:

It is a 4 legged spider with 2 servos in each leg



We needed to edit the design as it has 3 servos in each leg , so the new design is:



Software design

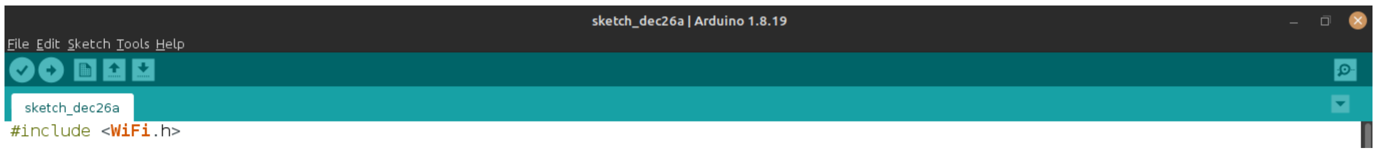
The software architecture of the spider robot involves a combination of embedded systems and control algorithms. The robot's microcontroller acts as the central processing unit, receiving commands and generating control signals for the individual servos.

The software program developed for the project is designed to control the movements of a quadruped robot using an ESP32 microcontroller board. The program utilizes the power of the ESP32 to establish a WiFi network as an access point, allowing another hexapods to connect with our robot. The program combines the functionality of the WiFi library for network communication and the Adafruit_PWMServoDriver library for servo control.

At its core, the software program consists of a series of functions that define different motion patterns for the quadruped robot. These functions utilize the Adafruit_PWMServoDriver library to control the servos connected to the robot's legs. By adjusting the servo angles, the functions enable the robot to perform various movements such as spinning, moving forward and backward, turning left and right, standing up, and shutting down.

Additionally, the program includes a localization function that utilizes the WiFi library to scan and analyze nearby WiFi networks for localization purposes. This function helps in determining the distance between the robots.

Here we'll illustrate our code:



```
#include <WiFi.h>
```

This line includes the WiFi library, which enables communication over a wireless network using the ESP32 board.

```
const char *ssid = "hexa_arwa";
const char *password = "12345678";
```

These lines define the SSID (network name) and password for the WiFi access point that will be created by the ESP32. Clients can connect to this access point to communicate with the ESP32.

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
```

These lines include the necessary libraries for controlling servos connected to the Adafruit PWM Servo Driver board. The Wire library is a communication library for I2C, and the Adafruit_PWMServoDriver library provides functions for controlling servos using the PWM signals.

```
Adafruit_PWMServoDriver pulseWidth = Adafruit_PWMServoDriver();
```

This line creates an instance of the Adafruit_PWMServoDriver class called pulseWidth, which will be used to control the servos.

```
#define servoMIN 150
#define servoMAX 600
```

These lines define the minimum and maximum servo positions. The values 150 and 600 represent the pulse widths in microseconds that correspond to the minimum and maximum angles of the servos.

```
void setup() {
  WiFi.softAP(ssid, password);
  pulseWidth.begin();
  pulseWidth.setPWMFreq(60);
  start_star();
}
```

The setup() function is called once when the microcontroller starts. In this code, it performs the following tasks:

- Sets up a soft access point (AP) using the provided SSID and password.
- Initializes the pulseWidth servo driver.
- Sets the PWM frequency of the servo driver to 60Hz.
- Calls the start_star() function, which sets the initial positions of the servos.

```
void setAngle(int servoNum,int ang){
    int pulse = map(ang ,0,180,servoMIN,servoMAX);
    pulseWidth.setPWM(servoNum, 0, pulse);
```

This function, setAngle(), takes a servo number (servoNum) and an angle (ang) and calculates the corresponding pulse width based on the provided minimum and maximum values. It then uses the pulseWidth servo driver to set the servo to the calculated pulse width.

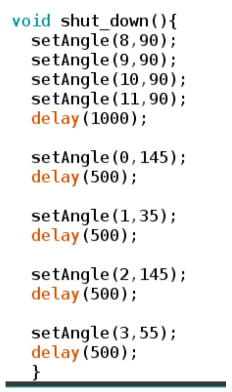
```
void start_star(){
  setAngle(0,90);
  setAngle(1,90);
  setAngle(2,90);
  setAngle(3,90);
  delay(1000);
  setAngle(8,0);
  setAngle(9,0);
  setAngle(10,0);
  setAngle(11,0);
  delay(1000);
}
```

This function, start_star(), sets the initial positions of the servos to create a star shape.

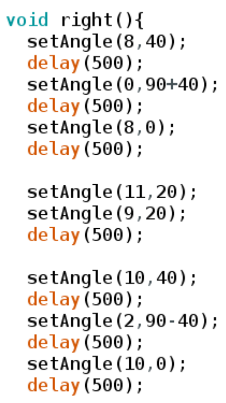
```
void spinning(){
//  this fn should be called before the spinning fn
//  start_star();

    setAngle(0,40+90);
    setAngle(8,30);
    setAngle(2,40+90);
    setAngle(10,30);
    delay(500);


    setAngle(8,0);
    setAngle(10,0);
    delay(1000);

    setAngle(3,20+90);
    setAngle(11,30);
    setAngle(1,40+90);
    setAngle(9,30);
    delay(500);
```

```
    setAngle(11,0);
    setAngle(9,0);
    delay(1000);


    setAngle(0,90);
    setAngle(1,90);
    setAngle(2,90);
    setAngle(3,90);
    delay(1000);
  }
```

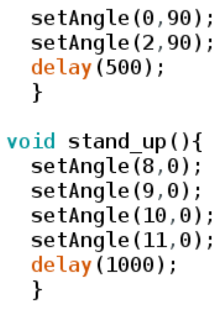
This function, spinning(), controls the servos to perform a spinning motion. It calls the setAngle() function multiple times with specific angles for each servo, creating a spinning effect.



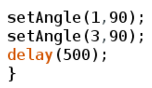
```
void shut_down(){
  setAngle(8,90);
  setAngle(9,90);
  setAngle(10,90);
  setAngle(11,90);
  delay(1000);

  setAngle(0,145);
  delay(500);

  setAngle(1,35);
  delay(500);

  setAngle(2,145);
  delay(500);

  setAngle(3,55);
  delay(500);
  }
```



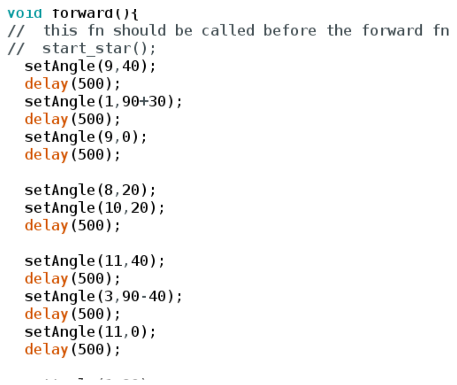
```
void right(){
  setAngle(8,40);
  delay(500);
  setAngle(0,90+40);
  delay(500);
  setAngle(8,0);
  delay(500);

  setAngle(11,20);
  setAngle(9,20);
  delay(500);

  setAngle(10,40);
  delay(500);
  setAngle(2,90-40);
  delay(500);
  setAngle(10,0);
  delay(500);
```



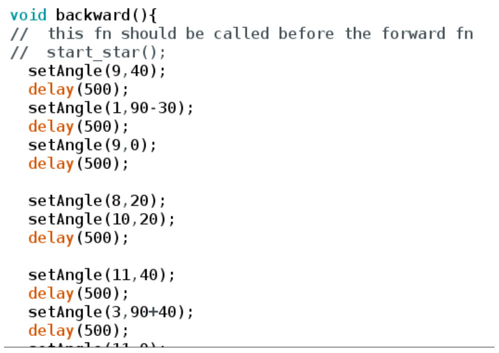
```
  setAngle(0,90);
  setAngle(2,90);
  delay(500);
  }

void stand_up(){
  setAngle(8,0);
  setAngle(9,0);
  setAngle(10,0);
  setAngle(11,0);
  delay(1000);
  }
```



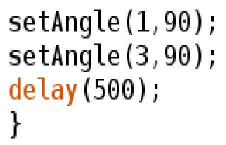
```
  setAngle(1,90);
  setAngle(3,90);
  delay(500);
  }
```



```
void forward(){
//  this fn should be called before the forward fn
//  start_star();
  setAngle(9,40);
  delay(500);
  setAngle(1,90+30);
  delay(500);
  setAngle(9,0);
  delay(500);

  setAngle(8,20);
  setAngle(10,20);
  delay(500);

  setAngle(11,40);
  delay(500);
  setAngle(3,90-40);
  delay(500);
  setAngle(11,0);
  delay(500);
```

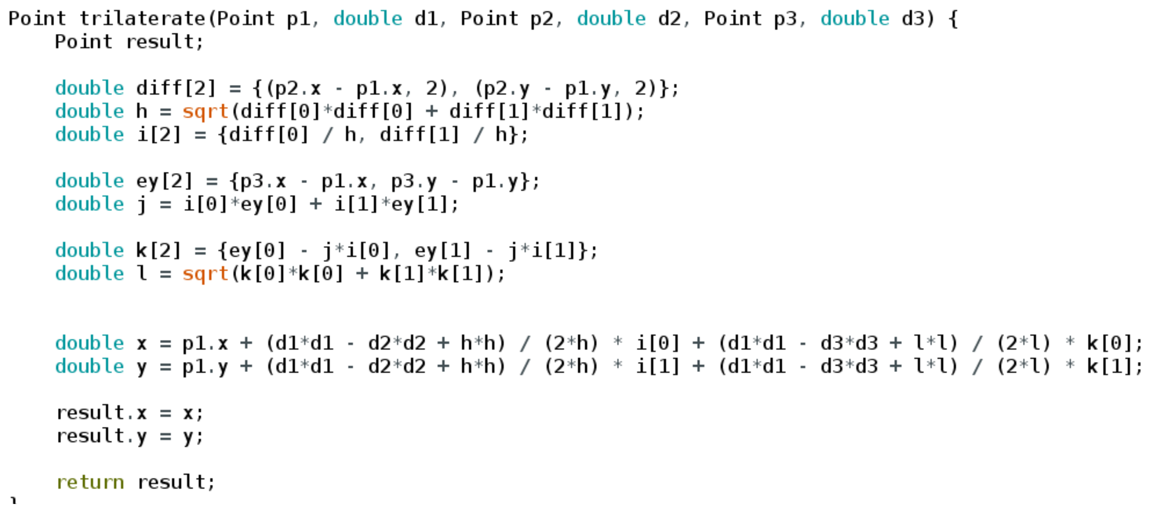


```
void backward(){
//  this fn should be called before the forward fn
//  start_star();
  setAngle(9,40);
  delay(500);
  setAngle(1,90-30);
  delay(500);
  setAngle(9,0);
  delay(500);

  setAngle(8,20);
  setAngle(10,20);
  delay(500);

  setAngle(11,40);
  delay(500);
  setAngle(3,90+40);
  delay(500);
```



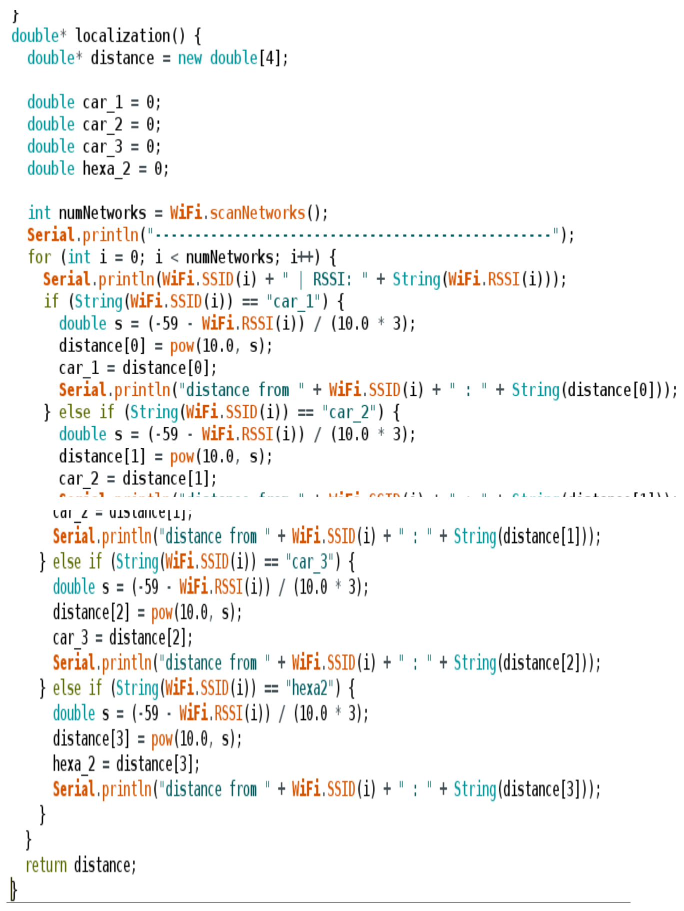
```
  setAngle(1,90);
  setAngle(3,90);
  delay(500);
  }
```

These functions (forward(), backward(), left(), right(), stand_up(), and shut_down()) define different motion patterns for the servos. Each function calls the setAngle() function with specific angles for the servos to achieve the desired motion.



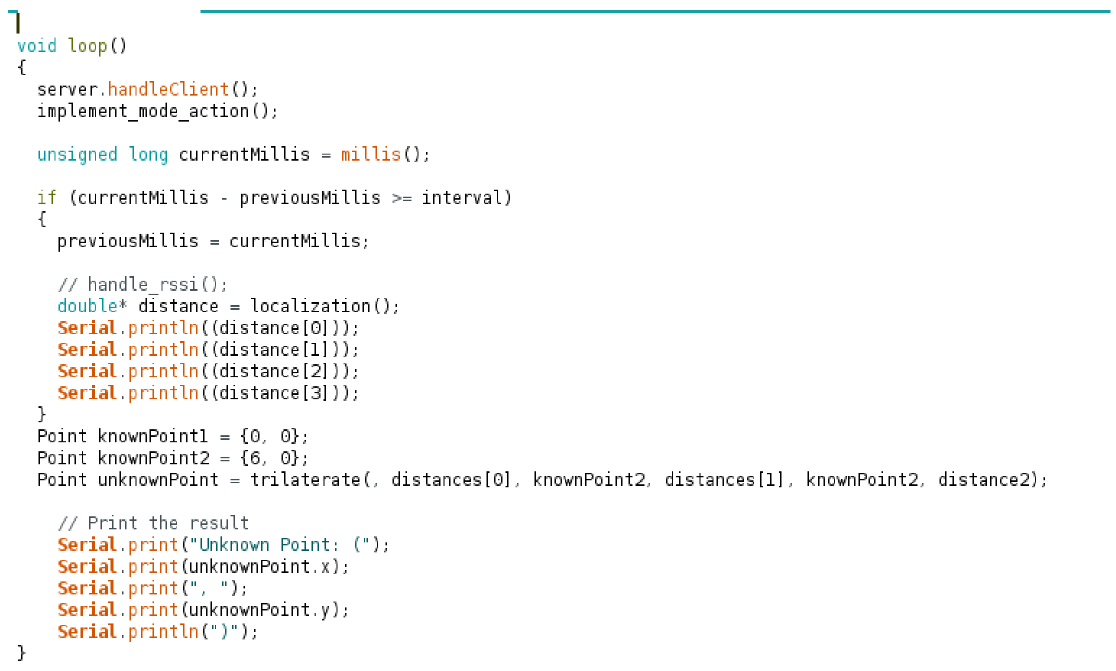
```
Point trilaterate(Point p1, double d1, Point p2, double d2, Point p3, double d3) {
    Point result;

    double diff[2] = {(p2.x - p1.x, 2), (p2.y - p1.y, 2)};
    double h = sqrt(diff[0]*diff[0] + diff[1]*diff[1]);
    double i[2] = {diff[0] / h, diff[1] / h};

    double ey[2] = {p3.x - p1.x, p3.y - p1.y};
    double j = i[0]*ey[0] + i[1]*ey[1];

    double k[2] = {ey[0] - j*i[0], ey[1] - j*i[1]};
    double l = sqrt(k[0]*k[0] + k[1]*k[1]);


    double x = p1.x + (d1*d1 - d2*d2 + h*h) / (2*h) * i[0] + (d1*d1 - d3*d3 + l*l) / (2*l) * k[0];
    double y = p1.y + (d1*d1 - d2*d2 + h*h) / (2*h) * i[1] + (d1*d1 - d3*d3 + l*l) / (2*l) * k[1];

    result.x = x;
    result.y = y;

    return result;
```

we have the trilaterate function that takes three known points (p1, p2, p3) and their corresponding distances (d1, d2, d3) as input. It calculates the coordinates of an unknown point using trilateration.



```
}
double* localization() {
  double* distance = new double[4];

  double car_1 = 0;
  double car_2 = 0;
  double car_3 = 0;
  double hexa_2 = 0;

  int numNetworks = WiFi.scanNetworks();
  Serial.println("-------------------------------------------------");
  for (int i = 0; i < numNetworks; i++) {
    Serial.println(WiFi.SSID(i) + " | RSSI: " + String(WiFi.RSSI(i)));
    if (String(WiFi.SSID(i)) == "car_1") {
      double s = (-59 - WiFi.RSSI(i)) / (10.0 * 3);
      distance[0] = pow(10.0, s);
      car_1 = distance[0];
      Serial.println("distance from " + WiFi.SSID(i) + " : " + String(distance[0]));
    } else if (String(WiFi.SSID(i)) == "car_2") {
      double s = (-59 - WiFi.RSSI(i)) / (10.0 * 3);
      distance[1] = pow(10.0, s);
      car_2 = distance[1];
      Serial.println("distance from " + WiFi.SSID(i) + " : " + String(distance[1]));
    } else if (String(WiFi.SSID(i)) == "car_3") {
      double s = (-59 - WiFi.RSSI(i)) / (10.0 * 3);
      distance[2] = pow(10.0, s);
      car_3 = distance[2];
      Serial.println("distance from " + WiFi.SSID(i) + " : " + String(distance[2]));
    } else if (String(WiFi.SSID(i)) == "hexa2") {
      double s = (-59 - WiFi.RSSI(i)) / (10.0 * 3);
      distance[3] = pow(10.0, s);
      hexa_2 = distance[3];
      Serial.println("distance from " + WiFi.SSID(i) + " : " + String(distance[3]));
    }
  }
  return distance;
}
```

The localization function performs WiFi signal strength-based localization. It scans for available WiFi networks and calculates the distances to specific network names (car_1, car_2, car_3, hexa2). The RSSI values are converted to distances using a formula, and the distances are stored in the distance array.



```
void loop()
{
  server.handleClient();
  implement_mode_action();

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis;

    // handle_rssi();
    double* distance = localization();
    Serial.println((distance[0]));
    Serial.println((distance[1]));
    Serial.println((distance[2]));
    Serial.println((distance[3]));
  }
  Point knownPoint1 = {0, 0};
  Point knownPoint2 = {6, 0};
  Point unknownPoint = trilaterate(, distances[0], knownPoint2, distances[1], knownPoint2, distance2);

    // Print the result
    Serial.print("Unknown Point: (");
    Serial.print(unknownPoint.x);
    Serial.print(", ");
    Serial.print(unknownPoint.y);
    Serial.println(")");
}
```

In the loop function, the localization function is called periodically to update the distances. Then, the trilaterate function is used to find the coordinates of the unknown point based on the updated distances. Finally, the coordinates of the unknown point are printed to the serial monitor.

Here is the full version of the code:
https://drive.google.com/drive/folders/1ejhLeLEIxs9r0OZ25jBakIr_uHUlT_1D?usp=sharing