Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Modular Robot Swarm

Project Engineering

Iwo Kania

Year 4

Atlantic Technological University Galway

Bachelor of Engineering (Honours) in Software and Electronic Engineering

2023/2024

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at the Atlantic Technological University Galway.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Acknowledgements

I want to thank my lecturers and ATU (Atlantic Technological University) staff for their help and feedback during the year. I also want to thank my classmates for their input and help throughout the year. I want to thank Brian O'Shea for being my supervisor for this project.

__Iwo Kania__

Summary

This project is a Modular Robot Swarm designed to carry out orders given by a single user to multiple robots at once. The purpose of this project is to be able to have cheap and easy to use robots that can be deployed anywhere in the world with minimal setup, only requiring two beacons and an internet connection. The inspiration behind this project was seeing a video of a concept robot swarm that could build bases on Mars by connecting and performing tasks together.

The Modular Robot Swarm is a network of drones developed to function seamlessly under a central control system accessible via the internet. It is designed to be easily scalable and adapt to different situations. The project's goal is to make each robot cost effective with as little hardware as possible so it can be expanded easily.

The Modular Robot Swarm works by the user sending orders to the webserver which the robots collect and execute. The webserver is hosted on an AWS (Amazon Web Services) server which communicates with the MongoDB database. The database holds the information for each robot and beacon.

The beacons get the distance between each other via Bluetooth Low Energy (BLE) to get the Relative Signal Strength (RSSI) as well as the RSSI between the beacons and robots. The robots calculate their co-ordinates based on the distance between it and the two beacons and triangulates its position. The robot sends and receives information from the webserver via HTTP requests. The information the robot receives is the location, name and ID of other robots, the distance between the two beacons and orders it needs to complete. The beacons act as a square barrier where the robots cannot cross, this is to ensure that the co-ordinates are accurate to prevent the robots from colliding with each other.

Contents

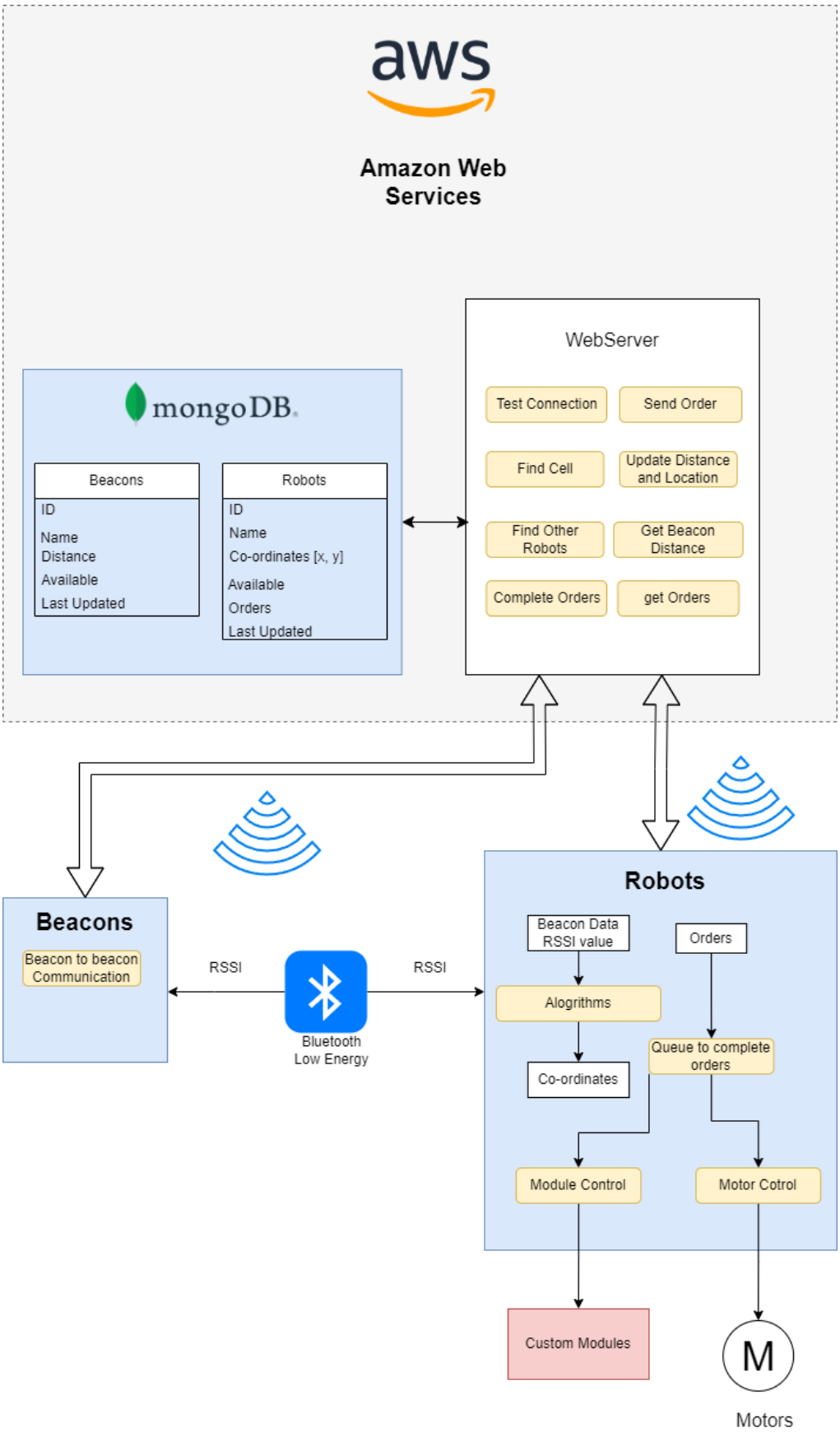
1. Introduction

The objective of the Modular Robot Swarm was to create a project based on the idea that industrial equipment is expensive and not easy to access in more remote areas around the world. I wanted to create a swarm that is cheap and adaptable where it can be used anywhere. I am also passionate about low-level programming and embedded systems and wanted to incorporate that into the project with FreeRTOS.

Since most modern machine networks use a central communication centre, I wanted to include one for my project with the use of Amazon Web Services. This provides a challenge to the project while highlighting skills and experience that can be useful in a wide variety of projects.

The project demonstrates a challenge in the development of an embedded project at a large scale with frequency HTTP and motor control. I had not had much experience with JavaScript and FreeRTOS before starting this project, I thought it would be a decent challenge applying newly learned concepts to the Modular Robot Swarm during the year.

1.1. Architecture Diagram



The project can be split into three parts: the webserver, the beacons, and the robots. The webserver is hosted on Amazon Web services alongside the MongoDB database. The webserver manages all the routes and communication between robots, beacons, and the database. The data the webserver collects is the beacons' name, ID, beacon to beacon distance, if it is available and the last time it was updated. The webserver also collects the robots' ID, name, its co-ordinates, orders, if its available and last time it was updated. The webserver tests connections of the robots and beacons and disconnects them from the network if they have not been updated for a certain time. The webserver finds available "cells" for the beacons and robots to occupy. The webserver can send the data of other robots to a robot that requested as well as send the beacon-to-beacon distance to a robot that requested it. It can send orders from users to the selected robot.

The beacons are used as a boundary where the robots cannot cross. They get the RSSI with BLE between each other and calculate the distance which then sends that data to the webserver and saved in that beacon's cell in the database.

The robots are the main component of the project. The robots are designed to communicate between it and webserver. To get its co-ordinates by getting the RSSI values from the two beacons and converting that to meters and using the webserver it gets the beacon-to-beacon distance. It periodically updates its location to the webserver as well as detects if it is too close to other robots. The robots get an array of orders from the webserver which then attempts to complete. It completes the orders by controlling motors, LEDs, and potentially custom modules.

2. Project Planning

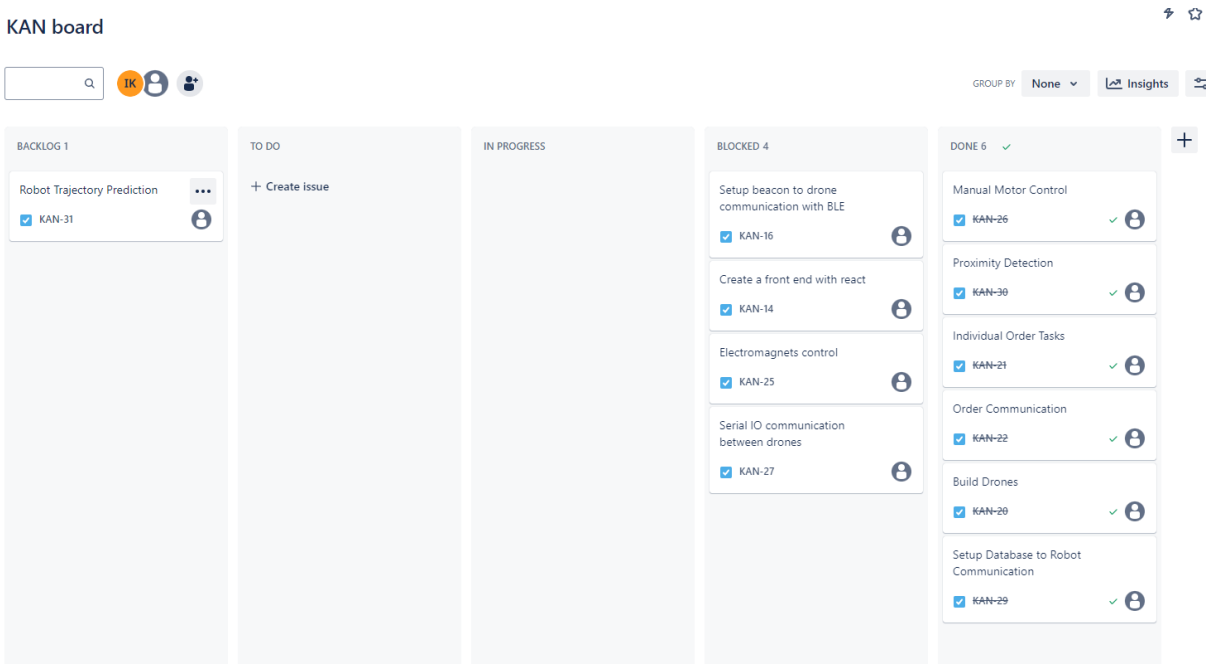


Figure 2.1: JIRA Kanban Board

For the project organization, I used a JIRA Kanban board. This kept me in relative focus while being an adaptable method to the ever – evolving project that changes in scope throughout the year. I used GitHub to throughout the project to keep track of the project and make it safer to make changes as any changes I make can be reverted if necessary.

3. Technology and Tools

3.1. Hardware

3.1.1. ESP32

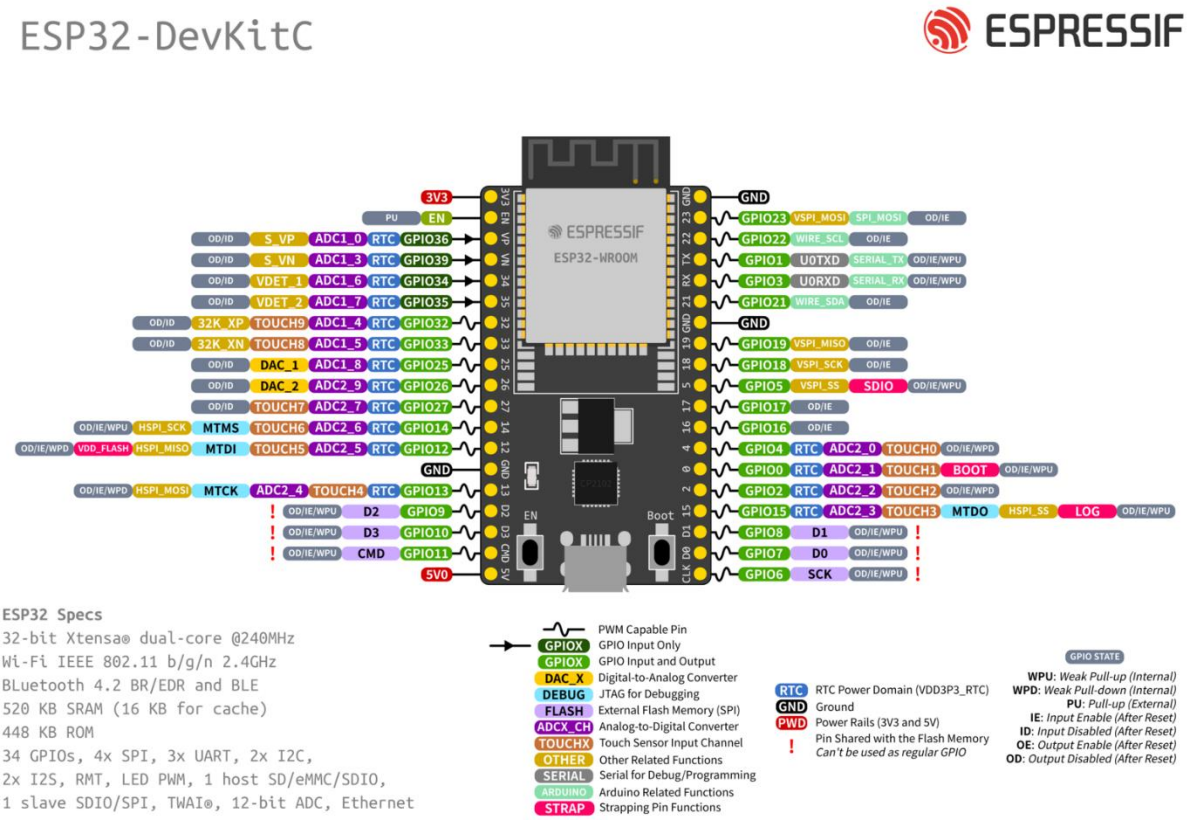


Figure 3.1: ESP32 Devkit C pinout Diagram [1]

The ESP32 by Espressif is a microcontroller I used for this project. The ESP32 is a highly integrated SoC (System on Chip), it contains a Wi-fi and Bluetooth hybrid chip [2] that was especially useful for my project as it uses BLE and HTTP requests. The ESP32 is a very affordable and fast microcontroller for the features it contains. It contains a large number of GPIO (General Purpose Input Output) pins that can be programmed to send a signal to control other aspects of the project.

3.1.2. LN298N

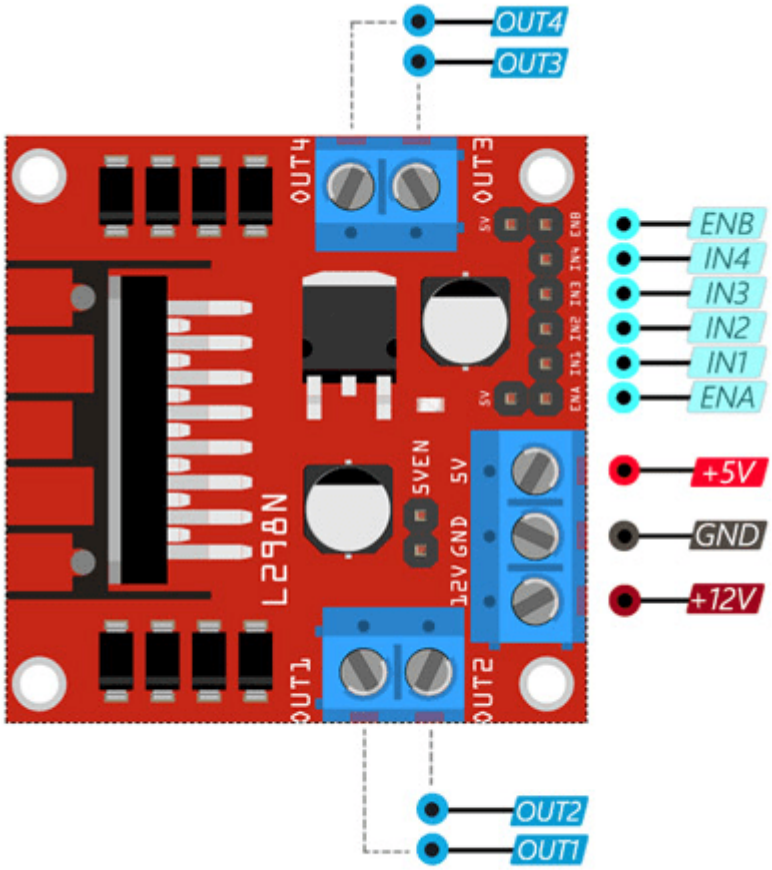


Figure 3.2: LN298N module pinout diagram [3]

The LB298N is a motor driver module that can be used to control 2 Stepper or DC motors. It uses an H-bridge to control direction and power supplied to the module; the IC (Integrated Circuit) is directly touching a heat sink as the H-Bridge produces a significant amount heat. I use it in my project to control DC motors on each robot.

3.1.3. DC Motor

The DC motor is a simple magnetic motor that I use as the main form of movement for my robots.

3.2. Software

3.2.1. Platform IO

The IDE for the robot and beacons is Platform IO. Platform IO is an IDE that is an extension on Visual Studio. Platform IO is a very versatile IDE as it supports many frameworks and development boards with integrated unit testing and Debugging tools [4]. Due to its ability to use the Arduino Framework on ESP32 boards instead of ESP-IDF framework as well as its easy configurability makes it an excellent choice for my project which includes both embedded software and webserver development.

3.2.2. AWS

Amazon Web Services Elastic Compute Cloud (AWS EC2) is a cloud-based server hosted by Amazon. EC2 is a technology that uses configurable and secure servers that are easily scalable by creating new instances. EC2 has elastic IP addresses meaning that each instance can be changed and moved with relative ease [5]. I use EC2 to host the webserver and it is the location of the database.

3.2.3. C/C++

C and C++ are one of the most popular and oldest programming languages. C++ is a variant of C that uses object-oriented programming. C++ is a low-level programming language which allows much more direct memory access, this makes C and C++ a great language for embedded systems. Many frameworks are written in C++ such as Arduino and ESP-IDF. This makes C/C++ a prefect language to program robots and beacons for my project.

3.2.4. FreeRTOS

FreeRTOS is an open-source Real Time Operating System for microcontrollers. Real Time Operating Systems is to perform tasks in clearly defined times [6], unlike in regular operating systems where they perform tasks as soon as possible. FreeRTOS works by completing tasks based on their availability and their set priority, tasks can also be set to only be executed when a communication object is called. FreeRTOS is used in my project because of its task-oriented execution making it easier to program a microcontroller with a large code base and being more power efficient.

3.2.5. JavaScript

JavaScript is a widely used programming language for webpages and web applications. It supports many frameworks used in webpages such as React, next.js, Angular, Express, etc. JavaScript is a high-level language that is user-friendly and forgivable. It easily supports the use of databases such as MongoDB as well as accessing information across the internet with fetch API. I use JavaScript in the webserver due to its easy server connectivity with the React framework.

3.2.6. Arduino

Arduino is a development platform for microcontrollers that are made specifically to be easy to use and learn. Arduinos are used in a wide variety of applications such as learning, prototyping and IoT [7]. It is an open-source platform with many libraries available based on the user's need. The Arduino framework is easy to use and code with as it is based on wiring [7]. I use the Arduino framework on the ESP32 with the use of the Arduino-ESP32 core [8]. I use the framework because of its ease of use and many available libraries that make developing easier such as ArduinoJSON to parse JSON strings received from the webserver as well as ArduinoBLE to be able to use the microcontroller's Bluetooth module as low energy.

3.2.7. MongoDB

MongoDB is a NoSQL database system which holds JSON objects. Each database can be split into collections where the objects can be stored. Its main advantage is that there is no set schema for the objects meaning that you can easily modify the objects' schema without needing to reset the database like in SQL databases [9]. I use MongoDB because of it being easily modified and easier to develop objects as the project continues as well as the fact its written in JSON format makes it easier to parse on the webserver and robots.

3.2.8. ChatGPT

ChatGPT is the most popular AI chat tools used in everything from helping find information, improve a paragraph of text, create and edit code or just to play around with. The ChatGPT is a generative AI language model, meaning it creates text based previous data it was trained on, any patterns or examples it was given, it uses that data and using specific algorithms to respond to user query [10]. I used ChatGPT to help as an assistant to help me find bugs in my code, give me example code that could help me or improve my code's reliability. Even though it couldn't help me debug my code sometimes, it gave me tips on how to find the problem with it.

4. Research

4.1. Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a module of the ESP32 which works differently to regular Bluetooth communication framework. BLE is designed to be energy efficient and send and receive relatively small amounts of data between devices. BLE is a very common communication technology used in IoT and medical devices [11]. BLE transfers information in a sort of Server – Client format, where there is a client called central and the server is called a peripheral. The central can be any device such as a phone or computer, the peripheral would be something like sensors or buttons. Peripherals can only connect to one device at a time and send information to the central while the central can connect to multiple peripherals and receive the information from the peripherals [12].

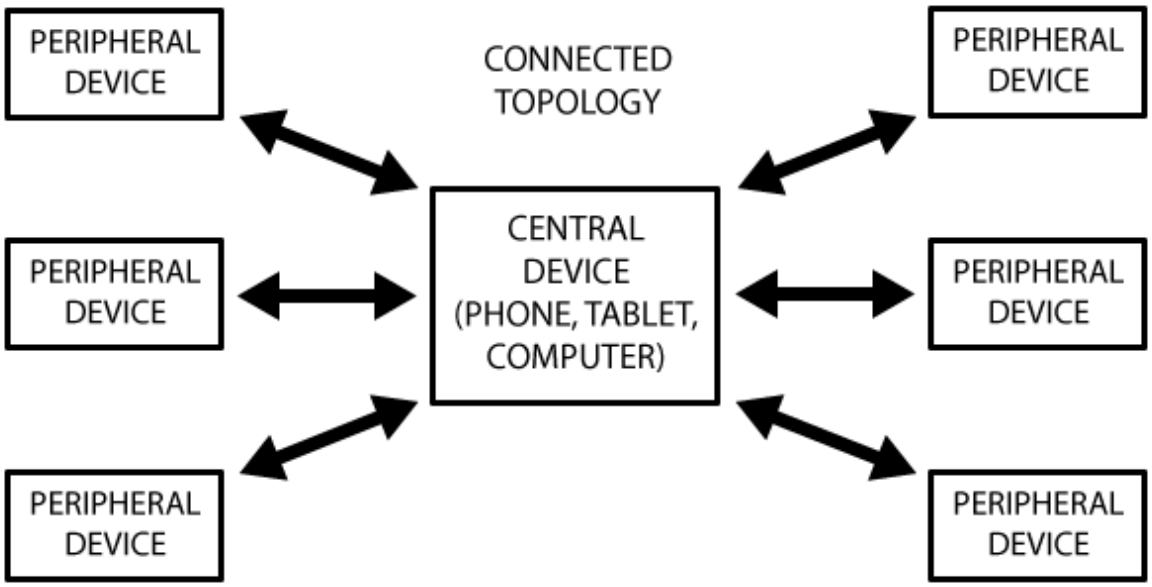


Figure 4.1: Diagram showing Central – Peripheral connection [12]

The information sent in a Generic Attribute profile (GATT), where each profile groups different services that indicate the purpose of the data for an application [13]. Each profile can have multiple services that contain multiple characteristics which send simple data packets to the central such as battery power levels.

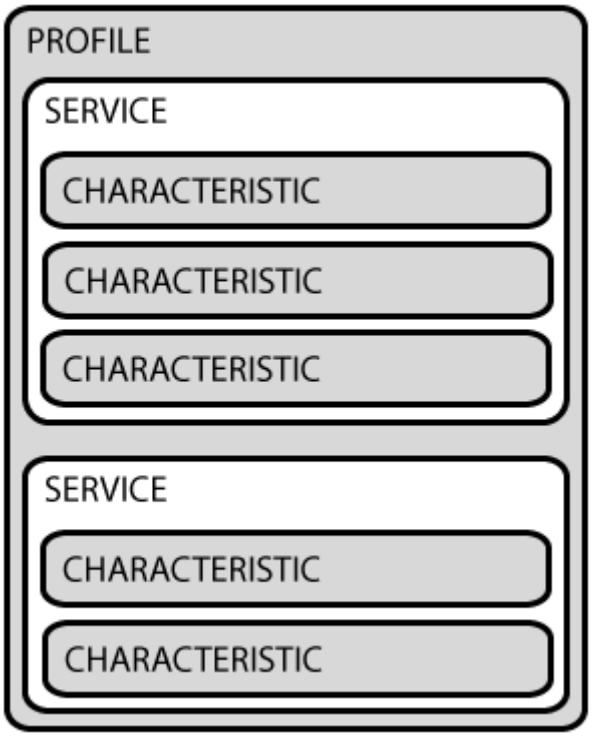


Figure 4.2: Diagram showing how each profile contains [12]

Characteristics each have UUIDs which can either be a preset 16-bit or a unique 128-bit number. Characteristics permissions you can set to each such as read, write, notify and broadcast.

4.2. Arduino String vs std::string

In the Arduino framework there is a data type called String which is an array of characters, this is a class that is based on the C++ std::string with notable differences. Arduino string and std::string dynamically allocates its data into the heap; this causes them to fragment if not the data is not managed correctly [14]. Arduino is missing much functionality that std::string contains to be more lightweight. The biggest advantage of the Arduino string in my project over std::string would be its easy conversion if string with numbers into a floating-point number or integer which a single function.

Std::string is from the C++ standard library, it shares many similarities as the Arduino string, such as memory allocation and converting to C string where the value of the string is stored defragmented and easier to reference or concatenate. The biggest advantage of std::string over Arduino string is its easy conversion of numerical values such as int, float and double into string to be concatenated.

4.3. Algorithms

This project required a few algorithms to operate different aspects of the robot. The RSSI to distance formula is used to convert the Relative Signal Strength Indicator (RSSI) and converts it into distance in meters, I used it to triangulate the distance between the two beacons and the robot.

$$\text{Distance} = 10 \wedge ((\text{Measured Power} - \text{RSSI})/(10 * N))$$

Figure 4.3: The RSSI to distance formula [15]

Measured Power refers to the measured RSSI value at 1 meter, RSSI is the current RSSI value and N is a constant set to take signal interference into account between two and four [15].

To triangulate the co-ordinates of the robot I needed to get the use the distance of each side of the triangle since I couldn't get the angles of the corners. To get the third co-ordinate I use the two known points as centers of circles and the beacon to robot distances as the radii of the circles. I then got the point of intercept between the two circles [16].

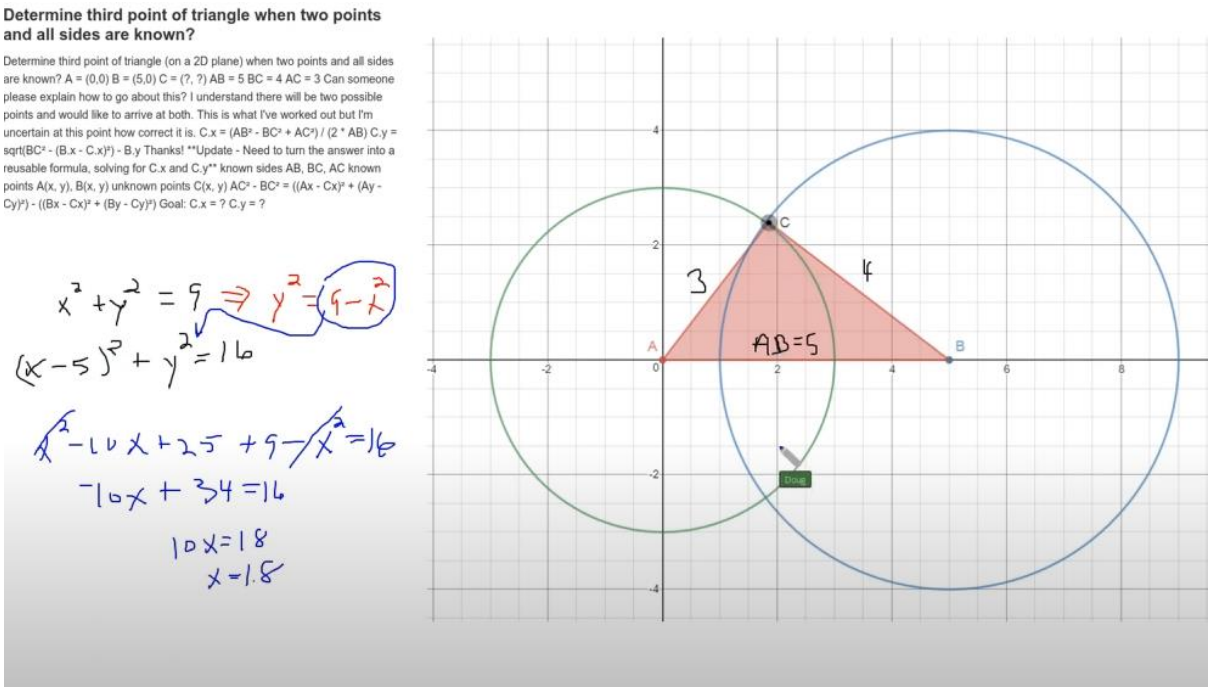


Figure 4.4: Example equation of getting the 3rd point of a triangle [16].

To get the distance between the robot and the edges of the boundaries and the robot, I used the formula distance to line formula by line being defined by two points [17].

If the line passes through two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ then the distance of (x_0, y_0) from the line is:[4]

$$\operatorname{distance}(P_1, P_2, (x_0, y_0)) = \frac{|(x_2 - x_1)(y_0 - y_1) - (x_0 - x_1)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}.$$

Figure 4.4 Formula of getting the distance of a line that is defined by two points [17].

4.4. Arduino Libraries

This project uses many libraries that have been developed for Arduino. The main two libraires I use are ArduinoJSON and HTTP Client. ArduinoJSON allows the program to parse JSON formatted strings, not to be confused with Arduino_JSON, ArduinoJSON is faster and easier to use, especially with the newest version 7 than Arduino_JSON. The ArduinoJSON library allows easy reading, writing and modifying of JSON data [18].

HTTP client allows the program to connect to IP addresses and send requests such as get, post and put. The libraries allow the return of data sent from the internet host and as well as response codes.

4.5. Platform IO

Platform IO is an IDE for many MCU's such as ESPs, Arduinos and STM's, it is also a plugin for various development software such as Visual Studio code and CLion. The Platform IO IDE is designed to be accessible to beginners and attractive for professionals [19], the IDE allows easy access to different frameworks for different boards as it works cross-platform and cross architecture and easily allows access to frameworks by just editing the platformio.ini file on the project [20].

4.6. FreeRTOS on ESP32

FreeRTOS works differently on different boards and frameworks as it's a close to machine system. While I was familiar with FreeRTOS on STM boards, using the operating system on the ESP32 and Arduino framework much was self-discovered. When creating tasks, on the STM boards, heap is allocated in words, while on ESP32, the heap is allocated in bytes. I also learned that the Arduino framework is already built on top of FreeRTOS, this happened when I was trying to use semaphores, I got an assert error. This error was since void setup () and void loop () are called from RTOS tasks.

5. Code

5.1. Robot

```
mrsHandle.BeaconfoundSemaphore = xSemaphoreCreateBinary();
mrsHandle.testConnectionSemaphore = xSemaphoreCreateBinary();
mrsHandle.getDistanceSemaphore = xSemaphoreCreateBinary();
mrsHandle.checkProximitySemaphore = xSemaphoreCreateBinary();
mrsHandle.distanceEvent = xEventGroupCreate();
mrsHandle.orderQueue = xQueueCreate(20, sizeof(mrsOrdersStruct_h));

xTaskCreatePinnedToCore(peripheralTask, "Peripheral Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 1, &mrsHandle.peripheral, 0);
xTaskCreatePinnedToCore(distanceTask, "Distance Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 2, &mrsHandle.distance, 0);
xTaskCreatePinnedToCore(findCellTask, "Find Cell Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 5, &mrsHandle.findCell, 0);
xTaskCreatePinnedToCore(testConnectionTask, "Test Connection Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 3, &mrsHandle.testConnection, 0);
xTaskCreatePinnedToCore(getBeaconDistanceTask, "Beacon-Beacon distance Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 5, &mrsHandle.getBeaconDistance, 0);
xTaskCreatePinnedToCore(updateLocationTask, "Update Location Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 4, &mrsHandle.updateLocation, 0);
xTaskCreatePinnedToCore(getOthersTask, "Get Other Robots Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 4, &mrsHandle.getOthers, 0);
xTaskCreatePinnedToCore(checkProximityTask, "Check Proximity Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 3, &mrsHandle.checkProximity, 1);
xTaskCreatePinnedToCore(getOrdersTask, "get orders Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 2, &mrsHandle.getOrders, 0);
xTaskCreatePinnedToCore(completeOrdersTask, "complete orders Task", 4 * KILOBYTE, NULL, configMAX_PRIORITIES - 2, &mrsHandle.completeOrders, 1);
```

Figure 5.1 Main task and communication object creation

The main files' job is to initialize BLE, GPIO pins, tasks and communication objects. mrsHandle is a structure that contains the reference objects to the semaphores and tasks, this is done to make sure that the variables will not be confused with similarly named variables. The function xTaskCreatePinnedToCore creates an RTOS task that can be set to a core in the microcontroller's CPU. Since the ESP32 has 2 CPU cores I can set them to either core 0 or core 1, I set all the logic to core 0 while the motor control to core 1. I did this so that motor control doesn't block the rest of the logic and sending requests to the webserver. The xTaskCreatePinnedToCore function parameters are the task function, name for the purpose of debugging, allocate space to the task in bytes, parameters for the function, priority (higher value is higher priority), the task handle for referencing the task and core to assign this task to.

The tasks can be split into three groups. BLE, webserver and motor tasks. The BLE tasks are used to scan for peripherals, get RSSI from beacons and calculate the distance from the RSSI and change the beacon the robot is scanning. The webserver tasks deal with all the communication between the robot and the webserver. It first finds an available cell in the database, the tasks also get the data of the beacons, other robots and the orders to complete from the webserver. Ther motor tasks deal with motor movement, they are called on a separate CPU core, they complete movement orders given by the webserver and check if they are near another robot or near the edge of the boundary.

```
//BLE Tasks
void peripheralTask(void * pvParameters);
void distanceTask(void * pvParameters);

//WebServer Tasks
void findCellTask(void * pvParameters);
void testConnectionTask(void * pvParameters);
void getOthersTask(void * pvParameters);
void updateLocationTask(void * pvParameters);
void getBeaconDistanceTask(void * pvParameters);
void getOrdersTask(void * pvParameters);

//Motor Tasks
void completeOrdersTask(void * pvParameters);
void checkProximityTask(void * pvParameters);
```

Figure 5.2 List of tasks in the robot

```
void peripheralTask(void * pvParameters) {
    for (;;) {
        //Serial.println("Peripheral Task");
        // check if a peripheral has been discovered
        taskVals.peripheral = BLE.available();

        if (taskVals.peripheral) {
            // discovered a peripheral
            Serial.println("Discovered a peripheral");
            Serial.println("-----------------------");

            // print address
            Serial.print("Address: ");
            Serial.println(taskVals.peripheral.address());
            if (taskVals.peripheral.localName() == "beacon1")
                taskVals.beacon1 = taskVals.peripheral.address();
            else if (taskVals.peripheral.localName() == "beacon2")
                taskVals.beacon2 = taskVals.peripheral.address();
            else
                Serial.println("Beacon name not found");

            BLE.stopScan();
```

Figure 5.3 Peripheral Task BLE scanning.

The peripheral task looks for any available BLE devices and checks if they go by the name “beacon1” or “beacon2” which if so then it saves the beacon’s address and stops scanning for other BLE devices to not have it be scanning in the background.

```
void getOthersTask(void * pvParameters) {
    for(;;) {
        Serial.printf("Get other Robots Task\r\n");
        //{"id":thisRobot.id}
        String tempResult = MRS_wifiGetJson("/findOthers", "{\"id\": " + std::to_string(thisRobot.id) + "}");
        if ((tempResult.toInt() != 404) || (tempResult.toInt() != 500)) {

            JsonDocument doc;
            DeserializationError error = deserializeJson(doc, tempResult);

            // Test if parsing fails.
            if (error) {
                Serial.print(F("deserializeJson() failed: "));
                Serial.println(error.f_str());
            } else {
                JsonArray robots = doc.as<JsonArray>();
                int i = 0;
                for(JsonObject robot : robots) {
                    std::string robotName = robot["name"].as<std::string>();
                    otherRobots[i].name = robotName;

                    int robotId = robot["id"].as<int>();
                    otherRobots[i].id = robotId;

                    JsonArray coords = robot["coords"];
                    otherRobots[i].coords[0] = coords[0].as<float>();
                    otherRobots[i].coords[1] = coords[1].as<float>();
```

Figure 5.4 Get other robots task.

This task uses the function ”MRS_wifiGetJson” to make a GET request to the webserver with route “/findOthers” the other parameter is a JSON string of id. I use JSON strings in this function so that I can easily change the variables I send to the webserver without having to change the code in the function much. The JsonDocument is a class of ArduinoJSON that can be used to contain variables to serialise or deserialise a string into. It then creates an array of JSON objects called robots which in the for loop parse each object into co-ordinates and the robot’s ID. I used ChatGPT to give me an example on how to parse a JSON object array but it gave me an example with outdated code where the parsed JSON would be stored in a StaticJsonDocument where I had to manually allocate memory to it, which is now deprecated.

```
void findCellTask(void * pvParameters) {
    for(;;) {
        Serial.println("FindCellTask");
        // {"name": "thisRobot.name", "coords":[thisRobot.coords[0], thisRobot.coords[1]]}
        std::string jsonPost = "{\"name\":\"" + thisRobot.name + "\", \"coords\":[" + std::to_string(thisRobot.coords[0]) + ", " +  std::to_string(thisRobot.coords[1]) + "]}";
        if (MRS_wifiPostJson("/findCell", jsonPost) == 200) {
            // {"name" : "thisRobot.name"}
            int tempResult = MRS_wifiGetJson("/getId", "{\"name\":\"" + thisRobot.name + "\"}").toInt();
            if ((tempResult != 404) || (tempResult != 500)){
                thisRobot.id = tempResult;
                vTaskSuspend(NULL);
            } else {
                vTaskDelay(1000 / portTICK_PERIOD_MS);
            }
        } else {
            vTaskDelay(1000 / portTICK_PERIOD_MS);
        }
    }
}
```

Figure 5.5 Find available cell in database Task.

This task is used to find any available cell in the database by sending a POST request to the webserver where the body is the JSON string. When found it looks for the id corresponding to its name and then finds it. If there was an error in any of their requests it repeats it after a second, if it was successful the task suspends itself until connection is lost and finding a new cell is needed.

```
void MRS_SetupConnection() {
        WiFi.begin(MRS_wifi.SSID, MRS_wifi.PASSWORD);
        Serial.println("Connecting");
    while(WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.print("Connected to WiFi network with IP Address: ");
    Serial.println(WiFi.localIP());
}
```

Figure 5.6 Setup Connection function.

There are three Wi-Fi functions, the initiation, POST and GET requests. The MRS_SetupConnection function come from Random Nerd Tutorials [21]. The function looks for an available internet host that matches the SSID and once connected WL_Connected is set to true which means that the robot is connected. The POST and GET request function are based on the Random Nerd Tutorials code [21].

```
if(WiFi.status()== WL_CONNECTED){
    WiFiClient client;
    HTTPClient http;

    url = MRS_wifi.serverName;
    url = url + page;

    http.begin(client, url.c_str());
    http.addHeader("Content-Type", "application/json");

    // Data to send with HTTP POST
    httpRequestData = postValue;
    // Send HTTP POST request
    httpResponseCode = http.POST(httpRequestData.c_str());
    Serial.printf("POST HTTP Response code: %d\n", httpResponseCode);

    http.end();
} else {
    Serial.println("WiFi Disconnected");
}
return httpResponseCode;
```

Figure 5.7 POST request function.

```
if (page == "/TestConnection") {
    std::string id = doc["id"];
    url = url + "?id=" + id + "&type=0";
} else if (page == "/getId") {
    std::string name = doc["name"];
    url = url + "?name=" + name + "&type=0";
} else if (page == "/getDistance");
else if (page == "/findOthers") {
    std::string id = doc["id"];
    url = url + "?id=" + id;
} else if (page == "/getOrders") {
    std::string id = doc["id"];
    url = url + "?id=" + id;
} else {
    Serial.printf("Page not found \r\n");
    return "404";
}

http.begin(url.c_str());

int httpResponseCode = http.GET();

if (httpResponseCode > 0) {
    Serial.print("GET HTTP Response code: ");
    Serial.println(httpResponseCode);
    returnJson = http.getString();
}
```

Figure 5.8 Part of the GET request function.

The POST request checks if it is connected to the Wi-Fi. HTTP. Begin starts a request to the given URL, the addHeader calls the request as a JSON, httpRequestData sends the value to be send to the request which HTTP. POST sends it as post request and returns the response which can a response code and JSON string. The Get Request instead of needing to create a header or send variables to the body, everything is handled in the URL.

```
float rssiToDistance(float fixedPower, float rssi, float environent) {
    float power = ((fixedPower + (float)abs(rssi)) / (10 * environent));
    float distance = pow(10, power);

    return distance;
}

/*
* coordBuf: Buffer pointing to Coordinate array
* b1r:      Distance between beacon 1 and robot
* b2r:      Distance between beacon 2 and robot
* bb:       Distance between  2 beacons
*/
void getRobotCoords(float* coordBuf, float b1r, float b2r, float bb) {
    float temp = pow(bb, 2) + pow(b1r, 2) - pow(b2r, 2);
    coordBuf[0] = temp / (2 * bb);
    coordBuf[1] = sqrt(fabs(pow(b1r, 2)) - pow(coordBuf[0], 2));
   if (b1r >= b2r)
        coordBuf[1] *= -1;
}

float distanceDiff(float coord1Buff[2], float coord2Buff[2]) {
    return sqrt(pow((coord2Buff[0] - coord1Buff[0]), 2) + pow((coord2Buff[1] - coord1Buff[1]), 2));
}

float edgeDistance(float line1Coords[2], float line2Coords[2], float robotCoords[2]) {
    float distance;
    float slope = (line2Coords[1] -  line1Coords[1]) / (line2Coords[0] -  line1Coords[0]);

    distance = (fabs(line1Coords[1] - slope * line1Coords[0] - robotCoords[1])) / (sqrt(1 + slope * slope));
    return distance;
}
```

Figure 5.9 Algorithm functions.

The Algorithms are codified from the algorithms mentioned before, I used math.h library to get methods like sqrt, which is square root, fabs which is absolute float value and pow which is power.

5.2. Beacon

```
Serial.print("RSSI: ");
Serial.println(peripheral.rssi());
char rssi[5];
sprintf(rssi, "%d", peripheral.rssi());

rssiValues[rssiCount] = peripheral.rssi();
rssiCount++;
if (rssiCount == RSSIAVGSIZE)
  rssiCount = 0;

for (int i = 0; i < RSSIAVGSIZE; i++) {
  if (rssiValues[i] == 0) {
    calcDistaceFlag = 0;
    break;
  } else {
    calcDistaceFlag = 1;
  }
}
if (calcDistaceFlag) {
  float avg = 0;
  for (int i = 0; i < RSSIAVGSIZE; i++)
    avg += rssiValues[i];
  avg = avg / RSSIAVGSIZE;
  float power = (((float)-44.6 + (float)abs(avg)) / ((float)10 * (float)4));
  distance = pow(10, power);
  Serial.printf("Distance: %.2fm\r\n", distance);
}
```

Figure 5.10 RSSI to distance code.

The beacon gets an array of RSSI values and waits until RSSIAVGSIZE of RSSI values have been collected before calculating the average RSSI value and converting it to distance.

```
    BLE.stopScan();
    adv_scan = 0;
    delay(50);
  }
} else {

  adv_scan = 1;
  BLE.scanForName(YOUBEACON.c_str());
  delay(10);
}
```

Figure 5.11 Switching scan to advertise.

Since the Bluetooth module doesn't allow scanning and advertising Bluetooth at the same time, I had to write it, so it iterates between scanning and advertising.

5.3. Webserver

```
// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/MRS', {
}).then(() => {
    console.log('Connected to MongoDB');
}).catch((error) => {
    console.error('MongoDB connection error:', error);
});

// Create a simple schema and model (replace with your own schema)
const robotSchema = new mongoose.Schema({
    id: Number,
    name: String,
    coords: [Number],
    available: Boolean,
    orders: [{direction: String, distance: Number, sent: Boolean}],
    lastUpdate: Number

});
const beaconSchema = new mongoose.Schema({
    id: Number,
    name: String,
    distance: Number,
    available: Boolean,
    lastUpdate: Number

});

const Robot = mongoose.model('Robots', robotSchema);
const Beacon = mongoose.model('Beacons', beaconSchema);
```

Figure 5.12 Setting up MongoDB connection and database with Mongoose.

The webserver connects to the local MongoDB to a database called MRS. These two schemas for robot and beacon will translate into JSON documents in the database. The available parameter means that that cell can be used by the robot or beacon. Last updated is the UNIX time in milliseconds which gets updated to the current time every time a route is being used. Orders is an array of objects used for robots to complete orders.

```
//Check if still connected
setInterval(async () => {
    try {
        let robots = await Robot.find({available: {$eq: 0}});
        let beacons = await Beacon.find({available: {$eq: 0}});

        robots.map(async robot => {
            if ((Date.now() - robot.lastUpdate) > 10000) {
                console.log("Lost connection with robot " + robot.id);
                await Robot.findOneAndUpdate({id: { $eq: robot.id}}, {available: 1})
            }
        })
        beacons.map(async beacon => {
            if ((Date.now() - beacon.lastUpdate) > 10000) {
                console.log("Lost connection beacon " + beacon.id);
                await Beacon.findOneAndUpdate({id: { $eq: beacon.id}}, {available: 1})
            }
        })
    } catch (error) {
        console.log("error 500: " + error.message)
        console.log(error.message);
    }
}, 5000)
```

Figure 5.13 Check connection for robots and beacons.

This checks if the beacons or robots are still connected, after 10 seconds it will set the available to 1 or true which means that the robots and beacons cannot use this cell anymore until they find a new cell. This runs every 5 seconds.

```
try {
    const robots = await Robot.find({available: { $eq: 1 }});
    res.json(robots);
    num = robots.length;
} catch (error) {
    res.status(500).json({ error: error.message });
    console.log(error.message);
}
console.log(num)
if(num == 0) {
    try {
        console.log("Available cells not found, creating new document")
        let setId = (await Robot.find()).length;
        await Robot.create({
                            id: setId,
                            name: req.body.name,
                            coords: [req.body.coords[0], req.body.coords[1]],
                            available: 0,
                            orders: [],
                            lastUpdate: Date.now()
                        });
    } catch (error) {
        res.status(500).json({ error: error.message });
        console.log(error.message);
    }
```

Figure 5.14 part of the find Cell route.

The find Cell route works by a robot or beacon sending a post request with its information and whether it's a beacon or robot. It first finds if there are any cells available with available equal true, if it is true then it gives its information to that cell and calls ownership over it. If there are no cells available, the webserver creates a new cell and then that robot or beacon calls ownership over it.

```
app.post('/sendOrder', async (req, res) => {
    const callerName = req.body.name;
    const callerOrders = req.body.orders;

    console.log("Caller Orders length:", callerOrders.length);
    try {
        const robot = await Robot.findOne({name: {$eq: callerName}})
        console.log("robot name: " + robot.name + " available: " + robot.available)
        if (robot != null) {
            if (!robot.available) {
                for (let i = 0; i < callerOrders.length; i++) {
                    await Robot.findOneAndUpdate({name: {$eq: callerName}}, {$push: {orders: {
                                                direction: callerOrders[i].direction,
                                                distance: callerOrders[i].distance,
                                                sent: 0}}});
                    console.log("Orders sent: " + callerOrders[i].direction  + " " + callerOrders[i].distance)
                }
                res.status(200).json({ message: "Orders sent successfully" });
            } else {
                console.log("Robot Not Available")
                res.status(400).json({ message: "Robot Not Available" });
            }
        } else {
            console.log("Robot Not found");
            res.status(404).json({ message: "Robot Not Found" });
        }
    } catch (error) {
        res.status(500).json({ error: error.message });
        console.log(error.message);
    }
});
```

Figure 5.15 Send Order Route.

This receives a JSON from the user from a POST request. The information sent is the robot's name the user wants to command and an array of orders the user wants the robot to complete. It first checks if the robot is connected to the webserver. It then finds and updates the document that corresponds to the robot which then pushes the order to the cell as well as update the last updated parameter to the current time.

```
app.get('/getOrders', async (req, res) => {
    try {
        let callerId = req.query.id;
        let robot = await Robot.findOne({id: {$eq: callerId}})
        await Robot.findOneAndUpdate({id: {$eq: callerId}}, {lastUpdate: Date.now()})
        let sendOrders = [];
        for (let i = 0; i < robot.orders.length; i++) {
            if (!robot.orders[i].sent) {

                console.log(robot.orders[i].distance + " " + robot.orders[i].direction)
                sendOrders.push({distance: robot.orders[i].distance, direction: robot.orders[i].direction});
                await Robot.findOneAndUpdate({id: {$eq: callerId}}, {lastUpdate: Date.now(),  $set: {[`orders.${i}.sent`]: true}})
            }
            console.log(sendOrders)
        }

        console.log("Orders sent:" + sendOrders);
        res.json(sendOrders)
    } catch (error) {
        res.status(500).json({ error: error.message });
        console.log(error.message);
    }
});
```

Figure 5.15 Get Order Route.

The get order route sends the order information to the robot that requested the data. The webserver checks if there are orders to send. Each order is pushed individually to an array sendOrders and as well as sets the sent parameter to true, so the order won't be sent again. It then sends the array to the robot.

6. Challenges

6.1. Co-ordinate Algorithm

A challenge I encountered early in the project was trying to get the coordinates of drones using the drones themselves as beacons. This mathematically translated to getting the coordinates of three points given one point and three lengths. This is unfortunately mathematically impossible, I had to reimagine the layout of the robots. I thought of using three beacons at first since that would create a boundless co-ordinate system where three points are known. From my supervisor's recommendation, I decided to use a two-beacon system where there is a boundary where the robots cannot move beyond of but requires less materials.

6.2. BLE Characteristics

I encountered challenges when trying to send characteristics from the beacons to the robot. The problem was that the robot was receiving the transmissions it just could not parse it. I thought instead of having the co-ordinate data be sent via BLE, I would send the data to a database which then the robot would collect. I did this since I would need a database for the robots anyway.

6.3. ESP32 Debugging

The ESP32 Devkit C3, the board I am using, doesn't have debugging firmware and it is hard to debug without it as I cannot create breakpoints without debugging and would need external hardware to be able to use debugging firmware on this board. This caused several issues, especially where there was near instant crashing on the board due to memory misallocation and was difficult to find. I had to create a lot of printing and checks to compensate for the lack of debugging functionality.

6.4. Assertion Failure

I had errors where whenever I tried to use semaphores, I got queue Semaphore errors and crashed the program. I couldn't find a solution on the internet nor ChatGPT could help me. After a few hours of trying different things, the solution was to initialise the semaphores before creating the tasks.

This was because unlike most other frameworks, the Arduino framework already has tasks scheduled as “void setup()” and “void loop()” functions are called inside a FreeRTOS task. The side-effect of this is that the placement of communication object initialisers must be before task creation. Whereas normally you would need to call task scheduler after creating tasks and initialising communication objects.

7. Conclusion

The Modular Robot Swarm project is a great step into drone swarm technology, as we increase our reliance on robots, swarm technology will play a pivotal part in it especially in transport and warehouse management. This project was a success in creating a working prototype that demonstrates the key technologies that could be used in drone swarms such as ESP32 microcontroller and AWS integration, as well as Bluetooth Low Energy as a distance indicator as well as using embedded systems development.

While not a complete success as there were some aspects of the project I wanted to add and improve on such a user-friendly front end and user-made custom orders, this was a great experience in project management and research.

8. References

[1] ESPRESSIF, "ESP32-DevKitC V4 Getting Started Guide," ESPRESSIF, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/hw-reference/esp32/get-started-devkitc.html. [Accessed 18 April 2024].

[2] ESPRESSIF, "Esperessif Products," [Online]. Available: https://www.espressif.com/en/products/socs/esp32. [Accessed 18 April 2024].

[3] HiBit, "How to use the L298N motor driver," HiBit, 30 January 2023. [Online]. Available: https://www.hibit.dev/posts/89/how-to-use-the-l298n-motor-driver-module. [Accessed 18 April 2024].

[4] PlatformIO, "Arduino IDE vs PlatformIO IDE," [Online]. Available: https://docs.platformio.org/en/latest/faq/arduino-vs-platformio.html. [Accessed 18 April 2024].

[5] I. Wigmore, "Amazon EC2 Instance," Tech Target, July 2021. [Online]. Available: https://www.techtarget.com/searchaws/definition/Amazon-EC2-instances. [Accessed 18 April 2024].

[6] Wikipedia, "Real-Time operating system," Wikipedia, 12 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/Real-time_operating_system. [Accessed 18 April 2024].

[7] Arduino, "What is Arduino?," Ardiuno, 05 February 2018. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 22 April 2024].

[8] ESPRESSIF, "arduino-esp32," ESPRESSIF, [Online]. Available: https://github.com/espressif/arduino-esp32?tab=readme-ov-file. [Accessed 22 April 2024].

[9] MongoDB, "What is NoSQL," [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed 22 April 2024].

[10] TechMobius, "Generative AI vs. LLM, What is the big difference?," LinkedIn, 28 November 2023. [Online]. Available: https://www.linkedin.com/pulse/generative-ai-vs-llm-what-big-difference-techmobius-6o6lc/. [Accessed 22 April 2024].

[11] GREAlpha, "THE DIFFERENCE BETWEEN BLUETOOTH AND BLUETOOTH LOW ENERGY," GREAlpha, [Online]. Available: https://www.grealpha.com/resources/articles/the-difference-between-bluetooth-and-bluetooth-low-energy. [Accessed 25 April 2024].

[12] Adafruit, "Introduction to Bluetooth Low Energy - GATT," 08 March 2024. [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. [Accessed 25 April 2024].

[13] Android, "Bluetooth Low Energy," [Online]. Available: https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview. [Accessed 26 April 2024].

[14] Majenko Technologies, "The Evils of Arduino Strings," 04 February 2016. [Online]. Available: https://hackingmajenkoblog.wordpress.com/2016/02/04/the-evils-of-arduino-strings/. [Accessed 26 April 2024].

[15] R. Shah, "Convert RSSI Value of the BLE Beacons to Meters — Best Solution," medium.com, 30 June 2021. [Online]. Available: https://medium.com/beingcoders/convert-rssi-value-of-the-ble-bluetooth-low-energy-beacons-to-meters-63259f307283. [Accessed 26 April 2024].

[16] M. F, "Determine third point of triangle when two points and all sides are known?," Wyzant, 21 March 2019. [Online]. Available: https://www.wyzant.com/resources/answers/601273/determine-third-point-of-triangle-when-two-points-and-all-sides-are-known. [Accessed 26 April 2024].

[17] Wikipedia, "Distance from a point to a line," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line. [Accessed 25 April 2024].

[18] ArduinoJson, "ArduinoJson," [Online]. Available: https://arduinojson.org. [Accessed 26 April 2024].

[19] PlatformIO, "What is PlatformIO?," [Online]. Available: https://docs.platformio.org/en/latest/what-is-platformio.html. [Accessed 26 April 2024].

[20] maxgerhardt, "Using github version of arduino-espressif32?," PlatformIO Community, 7 February 2024. [Online]. Available: https://community.platformio.org/t/using-github-version-of-arduino-espressif32/38325. [Accessed 26 April 2024].

[21] Random Nerd Tutorials, "ESP32 HTTP GET and HTTP POST with Arduino IDE (JSON, URL Encoded, Text)," Random Nerd Tutorials, [Online]. Available: https://randomnerdtutorials.com/esp32-http-get-post-arduino/. [Accessed 28 April 2024].

GitHub to the Project: https://github.com/Iwo1102/ModularRobotSwarm