თბილისის თავისუფალი უნივერსიტეტი

კომპიუტერული მეცნიერებების, მათემატიკისა და ინჟინერიის სკოლა (MACS[E])

ელექტრო და კომპიუტერული ინჟინერიის პროგრამა

ალექსანდრე სულაბერიძე

სენიორ პროექტი

რობოტების გროვებში ფორმაციისა და მწყობრი ქცევის მართვა.

ხელმძღვანელი: სულაბერიძე ზვიადი

თბილისი

2025

2. ანოტაცია/Annotation

The main idea of this project was to create a small swarm of identical robots, hexagonal in shape, with distance sensors on each side of the hexagon, and observe how we can make these robots complete high-level tasks with low-level information and action capability (e.g. distance to neighbors and simple movement). The project is meant as a proof of concept that a swarm of low-cost highly reproducible robots can complete difficult tasks,

Such as:

- **Transporting objects of different shapes and sizes in a factory:** This is the idea that the project is built around, a group of rovers with hexagonal plates on top that can connect to each other to form a continuous surface, which is then able to house various objects that will be transported by the rovers in unison.

- **Construction:** A lot of research today is based on the construction capability of robotic swarms and them being able to build structures using themselves as the basic building blocks. MIT Voxels and the MIT M-Block are the prime examples of this, and one of the inspirations to make a swarm-based project in the initial project planning phase.

- **Search & rescue robotic swarms:** Using robotic swarms that can divide and conquer large areas can greatly increase the efficiency of automated search and rescue systems, which are especially important in areas that are too dangerous for human rescue teams to enter.

My main goal was to create a working prototype of this system, which consisted of 3 to 6 rovers that each had 6 sensor modules consisting of a distance sensor, and an IR Transceiver, using which the robots would navigate their environment to complete higher-level tasks sent from the hub over WI-FI. For this proof of concept in particular, the higher-level task I chose was formation forming and flocking behavior using robotic rover swarms.

The advantage of robotic swarms is that each robot is low-cost and expendable, meaning that damage to one robot does not affect the system. Having the robots be cheap to make and make up the system as small modules, like ants, provides interesting insight about how we should tackle tasks efficiently using robotics.

ამ პროექტის ძირითადი იდეა იყო მცირე ზომის, იდენტური, ექვსკუთხა ფორმის რობოტების გუნდის (swarm) შექმნა, რომლებსაც ექვსივე მხარეს აქვთ მანძილის სენსორები და დაკვირვება იმაზე, თუ როგორ შეიძლება ამ რობოტებმა შეასრულონ მაღალი დონის ამოცანები დაბალი დონის ინფორმაციასა და მოქმედებებზე დაყრდნობით (მაგ., მეზობლებამდე მანძილი და მარტივი გადაადგილება). პროექტი არის “Proof of Concept" იმისა, რომ იაფი და მარტივად წარმოებადი რობოტების გუნდი შეიძლება გამოვიყენოთ რთული ამოცანების შესასრულებლად,

როგორიცაა:

- **ქარხანაში სხვადასხვა ფორმისა და ზომის ობიექტების გადაზიდვა:** სწორედ ამ იდეაზეა აგებული პროექტი — როვერების ჯგუფი, რომლებსაც ზემოთ ექვსკუთხა ფირფიტები აქვთ, რომლებიც ერთმანეთს უკავშირდებიან. ეს ფირფიტები ქმნიან უწყვეტ ზედაპირს, რომელზეც შესაძლებელია სხვადასხვა ობიექტების მოთავსება, რის შემდეგაც როვერები სინქრონულად გადაადგილდებიან ამ ობიექტების გადასაზიდად.

- **მშენებლობა:** თანამედროვე კვლევების დიდი ნაწილი ფოკუსირებულია რობოტული გუნდების (swarm) სამშენებლო უნარზე — იმაზე, რომ ისინი შეძლებენ სტრუქტურების აგებას საკუთარი თავის გამოყენებით, როგორც საბაზისო სამშენებლო ბლოკების. მისი ყველაზე აქტუალური მაგალითია MIT-ს M-Block-ები და Voxel-ები, რომლებიც მნიშვნელოვანი შტაგონების წყაროები იყვნენ პროექტის დაგეგმვის პროცესში.

- **სამშენებლო და სამაშველო რობოტული გუნდები:** რობოტული გუნდები, რომლებიც შეძლებენ დიდი ტერიტორიის განაწილებას და ინდივიდუალურად დამუშავებას, მნიშვნელოვნად გაზრდიან ავტომატიზირებული სამაშველო ოპერაციების ეფექტურობას, ეს განსაკუთრებით მნიშვნელოვანია ისეთ ადგილებში სადაც ადამიანებისგან დაკომპლექტებული სამაშველო გუნდებისთვის გადაადგილება საფრთხეს შეიცავს.
  ჩემი მთავარი მიზანი იყო სისტემის მუშა პროტოტიპის შექმნა, რომელიც შედგებოდა 3-იდან 6-მდე როვერისგან. თითოეული მათგანი მოიცავს 6 სენსორულ მოდულს, რომლებიც მოიცავენ მანძილის სენსორს და ინფრაწითელ მიმღებ-გამგზავნს (IR Transceiver). ამ მოდულების მეშვეობით რობოტები გარემოში გზას იკვლევენ და ასრულებენ მაღალი დონის ამოცანებს, რომლებიც WI-FI-ს მეშვეობით იგზავნება ცენტრალური ჰაბიდან. ამ “Proof of Concept-ისთვის” არჩეული მაღალი დონის ამოცანა იყო ფორმაციების (formation) ჩამოყალიბება და "ფლოქინგი" — ერთობლივი გადაადგილების ქცევა რობოტულ გუნდებში.

  რობოტული გუნდების უპირატესობა იმაში მდგომარეობს, რომ თითოეული რობოტი იაფია და მარტივად გამოცვლადი — ერთი რობოტის დაზიანება მთლიან სისტემას არ ანგრევს. ასეთი მოდულარული მიდგომა, სადაც რობოტები თითქოს ჭიანჭველებივით მცირე და შეთანხმებულ ნაწილებად ქმნიან სისტემას, გვაძლევს საინტერესო ხედვას იმაზე, თუ როგორ უნდა მივუდგეთ ამოცანების ეფექტიანად გადაჭრას რობოტიკის გამოყენებით.

3. Table of Contents:

4. List of Formulas, Graphs, and Pictures

#1 - *Initial Block Diagram*

#2 & #3 – Lower-Layer *CAD + Exploded View*

#4 & #5 – Middle-Layer *CAD + Exploded View*

#6 – Top-Layer CAD

#7 & #8 – Sensor Module CAD – Top & Bot Side

#9 & 10 – Full Assembly CAD + Exploded View

#11 - ESP32-S3-DevkitC-1

#12 – Main PCB Layout (All Layers)

#13 – Secondary PCB (Both Layers)

#14 – Network Architecture Diagram

#15 – Robot Software Architecture Diagram

#16 – Idle Mode Diagram

#17 – Line Mode Diagram

#18 – Polygon Mode Diagram

#19 – Three Robots Side by Side

5. Introduction

Moving forward after my Junior Project, it proved to be quite difficult to choose what I wanted to do for my Senior/Bachelor's project. I knew I wanted to do a project that required me to build my own PCB from scratch, I wanted to use an ESP32 MCU since I hadn't done much of that before, I wanted to learn how to use ROS (which I opted out of at the end, next project though...) and, as per usual, I wanted my project to be visually pleasing and interesting to the non-engineer eye.

I went through a couple of iterations before stopping at my final idea, one of the more interesting ones was a wheel robot that could traverse terrain and move autonomously, its main source of movement being a movable weight inside of it that shifted its center of mass to the direction it needed to move. I decided not to follow through with this idea because of the mechanical complexity.

My final idea, the robotic swarm project, was mainly inspired by the Disney film "Big Hero 6", in which the main character has a swarm of "Microbots" that can build complex structures and combine to perform a lot of tasks, all the while the individual robots are only able to complete the simplest of movements.

While the relevance and potential applications of this project have been discussed in detail in the annotation, it is worth mentioning that such a system could have diverse uses in fields like security, defense, medical, and industrial automation, among others, depending on the choice of the end effector and specific requirements.

Note:

Sentences written using ~~strikethrough~~ represent previous versions of the project that I choose to keep within the report.

6. The Problem Statement & The Proposed Solution

6.1. Research Conducted Related to the Project Topic

Existing Technologies and Market in the Field:

In recent years, there has been a growing interest in the application of robotic swarms, particularly within academic research. While swarm algorithms—such as those based on flocking, formation control, and distributed consensus—have been under development for decades in the field of software and control theory, the hardware implementation of swarm robotics has gained momentum more recently, particularly over the past decade.

Despite this progress, there are still relatively few notable swarm robotics projects deployed in industry, especially when it comes to ground-based or general-purpose swarm systems. The most prominent exception is the use of drone swarms, which have seen applications in defense, entertainment, and coordinated surveying tasks.

The majority of research and development in robotic swarms remains concentrated in academic institutions with strong robotics programs, such as Carnegie Mellon University (CMU), the Massachusetts Institute of Technology (MIT), ETH Zurich, and others. These institutions have led key innovations in both modular robotics and swarm coordination, such as MIT's M-Blocks and Kilobot projects, which serve as important references for this work.

Challenges in the Field:

Swarm robotics faces several key challenges. **Scalability** is difficult, as algorithms that work for small groups often fail in larger swarms due to communication and coordination issues. **Localization** in GPS-denied or dynamic environments is another major hurdle. **Energy efficiency** limits robot capabilities, especially for small mobile units. Hardware remains a challenge as well—building **low-cost, reproducible platforms** is not yet standardized.

Future Development Perspectives in the Field:

Swarm robotics is a rapidly advancing field with growing research in areas like construction, transportation, search and rescue, defense, and entertainment. As hardware becomes more standardized and affordable, the potential for real-world applications will expand significantly. In the future, we may see factories operated by swarms, structures built by self-organizing robots, and widespread use of swarms in environments too complex or dangerous for humans.

Problem Statement and Novelty:

This project aims to serve as a proof of concept that robotic swarms can exhibit complex behaviors—specifically formation and flocking—using only low-level information, such as the distance to neighboring robots, and low-level actions, such as simple movement in a 2D plane.

The novelty lies in demonstrating that **emergent coordination and collective intelligence** can arise from a group of **simple, identical, low-cost robots** without relying on global positioning systems, centralized control, or high-level sensing. This approach supports the idea that minimalism in hardware and communication can still lead to effective swarm behavior, making the system scalable, robust, and accessible.

Technical Objectives and Requirements:

- **Design and build a low-cost, modular rover** with basic movement and sensing capabilities to serve as the fundamental unit of the swarm system.
- **Develop a custom PCB** integrating an ESP32 System-on-Module (SoM), required peripherals, sensor connectors, and a power management system to support autonomous operation.
- **Create a simple, robust chassis** that is easy to manufacture, assemble, and replicate.
- **Establish a wireless communication network** using Wi-Fi, enabling 3 to 6 rovers to receive commands from a central hub (e.g., a PC) and act in coordination.
- **Implement and test swarm algorithms** that enable decentralized decision-making based on low-level sensor inputs, focusing on formation control and flocking behavior.

Relevance and Potential Applications:

Swarm robotics holds significant promise in domains where scalability, fault tolerance, and adaptability are critical. Applications include **automated warehouses**, **search and rescue operations** in hazardous environments, **construction of modular structures**, **environmental monitoring**, and **collaborative transportation** of irregular objects. The decentralized nature of swarms makes them ideal for situations where individual robot failure must not compromise the overall task.

Commercialization Potential:

As hardware becomes cheaper and more standardized, swarm-based systems could enter the commercial market in logistics, manufacturing, and infrastructure maintenance. Low-cost, modular swarm robots can reduce labor and operational costs, provide flexibility in reconfigurable tasks, and enable scalable automation solutions. With growing interest from industries in autonomous systems, the commercialization potential of robust, swarm-based platforms is strong—especially in niche environments where traditional robots fall short.

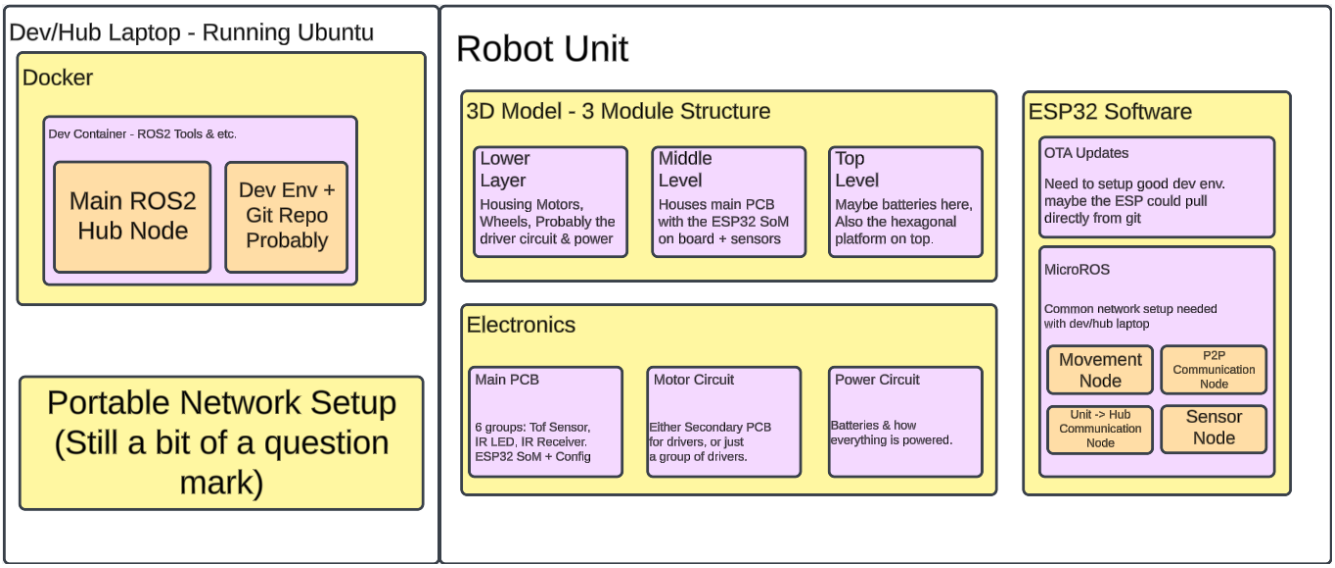
6.2. The Technical Side of the Project

The final version of the project mostly follows the proposed initial block diagram; the main differences are that I opted out of using ROS and MicroROS because of the time constraints and the learning curve, which eliminated the need for the hub to run Ubuntu, so instead the robot is

programmed using the PlatformIO IDE extension in VS Code on a computer running regular windows.

The technical discussion of the project will be divided into 3 parts:

- The mechanical design of the rovers.
- The electronics & control, particularly the PCB design.
- The software.



Picture #1 – Initial Block Diagram

6.2.1. General Overview

The main node of the system is the individual rover. The rover consists of its 3D printed 3-part chassis, 6 sensor modules, 3 stepper motors, 3 omni-directional wheels, the main PCB that handles all the power distribution and control, a secondary PCB that handles the motor control, and a battery pack that houses 3 LiPo batteries to provide 12 volts of power. Each of these will be discussed in detail.

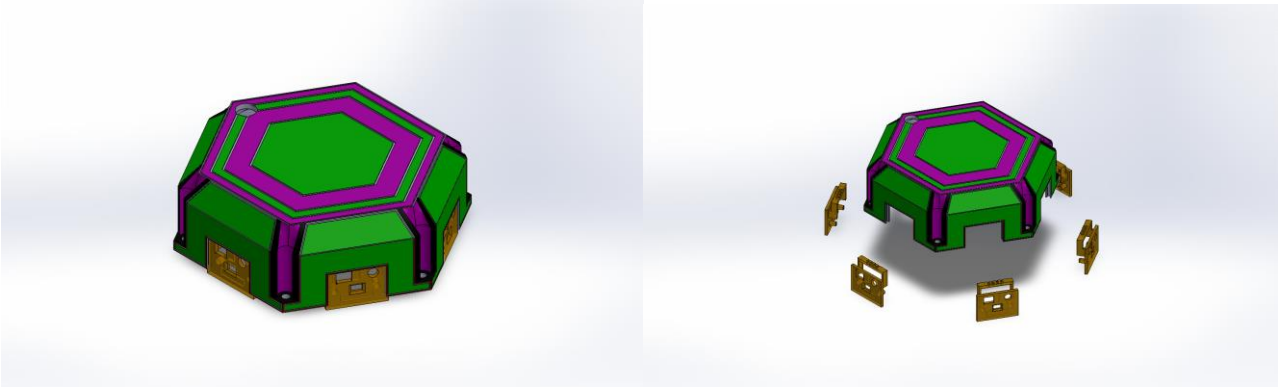
6.2.2. Mechanical Design of the Rover

6.2.2.1. Lower Layer

Picture #2 & #3 – Lower-Layer CAD + Exploded

The lower and most important part of the chassis consists of 3 stepper motor slots, 2 sets of pedestals for the PCBs, airflow holes at the bottom, towards which the radiators of the stepper motor drivers are pointed, and shallow spots, in which the sensor modules are oriented. The part also features screw holes for the stepper motors, as well as ones for the middle-layer connection.

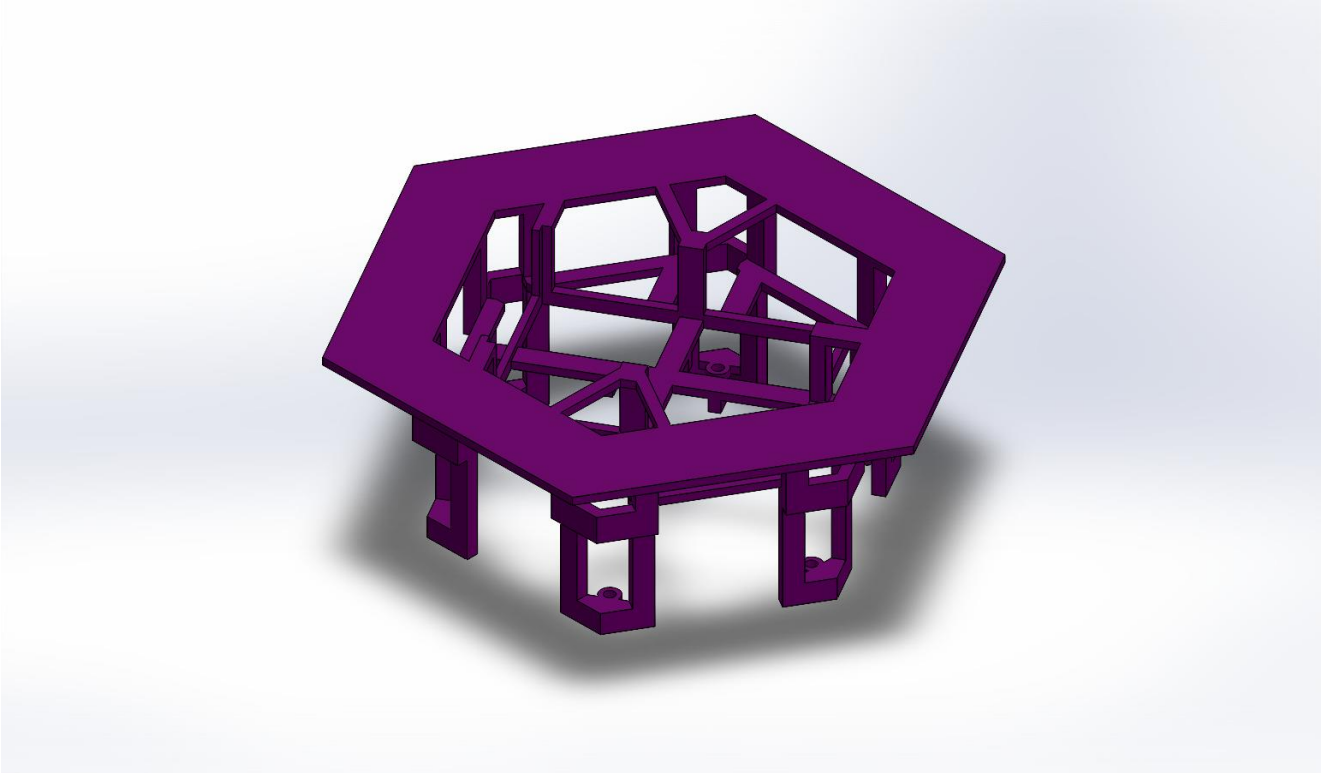
6.2.2.2. Middle Layer



Picture #4 & #5 – Middle-Layer CAD + Exploded

The middle part of the chassis is a shell-like cover that consists of 6 slots for the sensor modules, screw holes to connect to the bottom-layer and a hole that feeds power from the battery pack on top to the inner electronics of the robot.

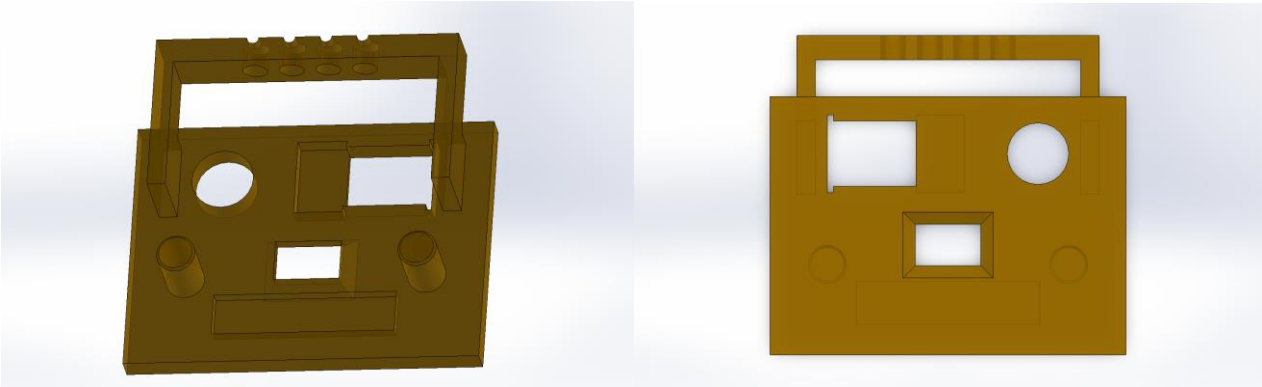
6.2.2.3. Top Layer



Picture #6 – Top-Layer CAD

The top part of the chassis is a single part that rests on top of the closed shell of the rover, connecting to the same bolt holes that connect the middle and the bottom layer together. It contains a slot for the battery holder of the robot that completes the hexagonal surface on which it is meant to transport the previously discussed objects. It is designed in a way that allows screwdriver access to the bolt holes, as previous designs proved to make the bolts impossible to drive through their respective slots

6.2.2.4. Sensor Module



Picture #7 & #8 – Sensor Module CAD – Top & Bot Side

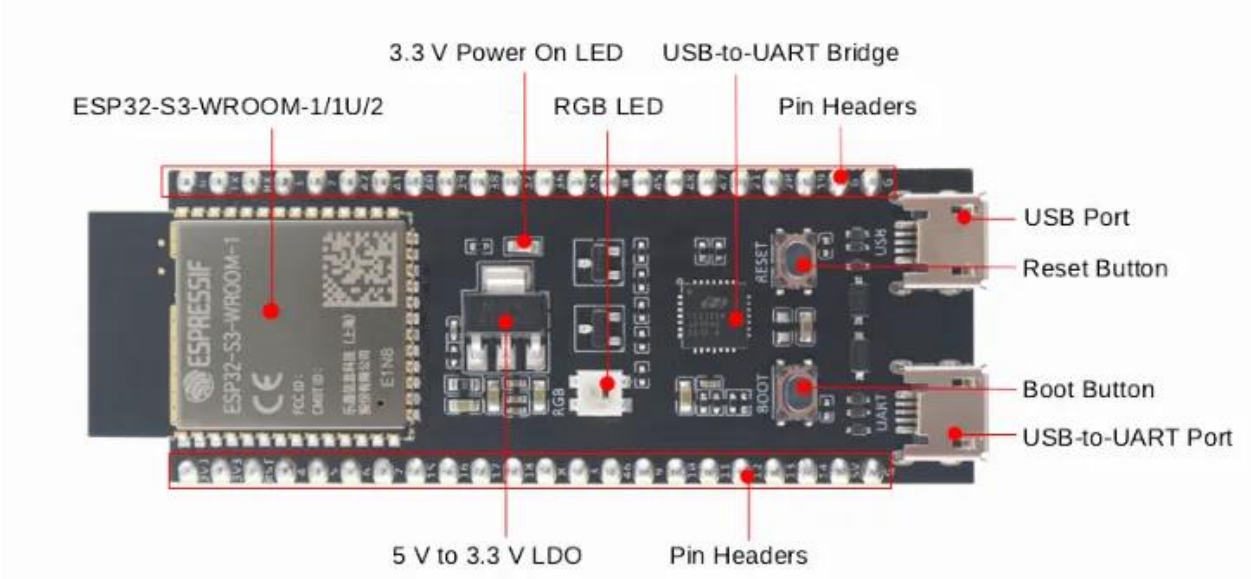
The sensor module has 3 slots, the bottom one is intended for the ToF distance sensor, and the 2 on the top are intended to house an IR LED and IR receiver, all of which will be discussed in detail later. The protrusion on the top is meant to organize the connectors of the IR transceiver to simplify the soldering process, as without it, it's very messy and prone to shorting together.

Picture #9 & #10 – Full Assembly CAD + Exploded

6.2.3. Electronics & Control

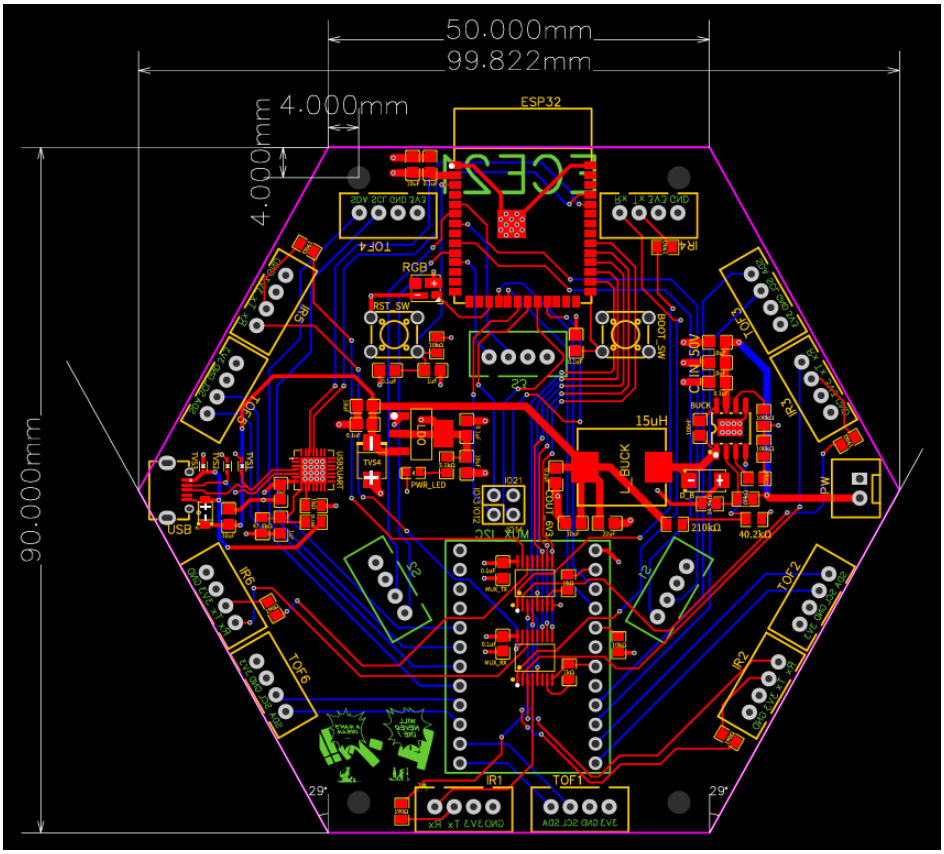
6.2.3.1. Main PCB Design

The brains of the rover is a 4-layer PCB that is home to an ESP32-S3 SoM. The ESP is responsible for all the computation, as well as communication with the hub, and is set up on the PCB as it would be on a development board. The PCB is based on the ESP32-S3-DevKitC-1 development board, containing most of its functionality with a few tweaks.



Picture #11 – ESP32-S3-DevkitC-1

The following will be a detailed dissection of this PCB, which will not include the details already included within the documentation of the ESP32-S3-DevKitC-1. The schematic and PCB layout can be viewed in detail at the GitHub link provided in the appendix.



Picture #12 – Main PCB Layout (All Layers)

The four layers of the PCB include 2 layers dedicated solely to power, the GND and VCC layers, that connect to each of the pins and pads they are intended to connect to using vias.

The SMD components of the board are placed on the top layer, while the THT components are organized around both the top and bottom layers. The board also features 4 screw holes that are placed directly on the pedestals mentioned above in the CAD section, and it's also worth to note that the antenna of the ESP32-S3 SoM sticks out of the board contour to safeguard it from some of the unwanted electromagnetic noise it may experience. The board also features a small block of PCB art on the bottom layer.

Starting with the ESP32-S3-DevkitC-1 reference, the board features an ESP32 SoM, an RST and BOOT button, a USB2UART along with a USB connector, an LDO to drop the 5V input to 3.3V and an RGB LED tied directly to one of the ESP pins.

What's different from the reference:

- The PCB does not include auto-programming transistors that allow you to flash code onto it without manually pressing the boot button, because it is meant to be programmed OTA (Over the air).
- It has a couple of extra TVS Diodes here and there to protect it from even more transients that will arise from the power being coupled to 3 stepper motors and their drivers.
- The separate ESP-USB connector is omitted in favor of the regular UART communication capabilities of the ESP32-S3 so as not to complicate it.
- Besides the pins that go straight to the connectors discussed after this, we have 4 breakout GPIO pins that don't have a set purpose.

PCB specific modules:

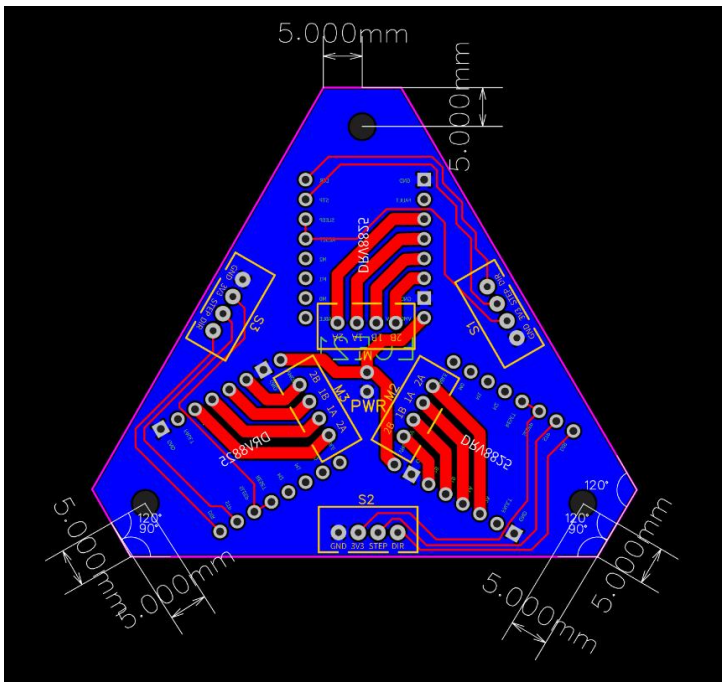
- The PCB features a MP1584EN buck converter that converts the 12V battery input to 5 volts and feeds it into the LDO. The buck converter circuit is taken straight from the MP1584EN datasheet.
- On the top layer we have 6 pairs of 2.54mm 4Pin connectors. The pair consists of a connector the IR Transceiver (Rx, Tx, VCC, GND) and a connector to the ToF Sensor (SDA, SCL, VCC, GND). The connectors are oriented in a way the support the placement of the sensor modules.
- The board features an I2C multiplexer for the 6 ToF Sensors, and 2 analog multiplexers, one for the IR LEDs and the other for the IR receivers.
- On the bottom layer, we have 3 connectors that feed into the secondary PCB, these connectors include logic level power, and the STEP and DIR signals needed to drive the stepper motors.

All of this is discussed in even more detail within the schematics-and-pcb directory in the GitHub repository provided in the appendix.

Some important considerations made when designing this PCB were:

- Routing the USB traces as differential pairs.
- Not routing signal traces beneath the buck converter inductor.
- The widths of the power traces so they safely carry the current that they are supposed to carry.
- Vias beneath IC thermal pads for better heat dissipation.
- Power plane connections with thermal pads to limit heat dissipation so soldering is easier.

6.2.3.2. Secondary PCB Design



Picture #13 – Secondary PCB (Both Layers)

The secondary PCB is much simpler and is only meant as a bridge between the main PCB and the 3 stepper motors.

The board contains 3 DRV8825 drivers, 2 pins for a big capacitor and 12V power input fed from the top PCB battery connector, 3 bolt holes that are placed on the pedestals mentioned in the CAD section, and 6 connectors, 3 of which connect to the main PCB, and the other 3 directly to the stepper motors.

One of the important considerations made for this PCB is the fact that most of the traces are not signal traces and are power traces, which is why they are doubled on both layers and significantly wider than the signal traces.

6.2.3.3. Sensor Module Design

The sensor module consists of a VL53L0X time of flight sensor, an IR333 LED, and a TSOP38238 IR receiver. The module is organized onto the CAD model in a way that it has a pair of 4Pin connectors that connect straight to the main PCB.

The IR transceiver is used along with the ESP32-S3 RMT (Remote Control Peripheral) peripheral to allow the robots to pass their unique ID onto their neighbor, which allows them to establish an ESP-NOW connection.

6.2.4. Software

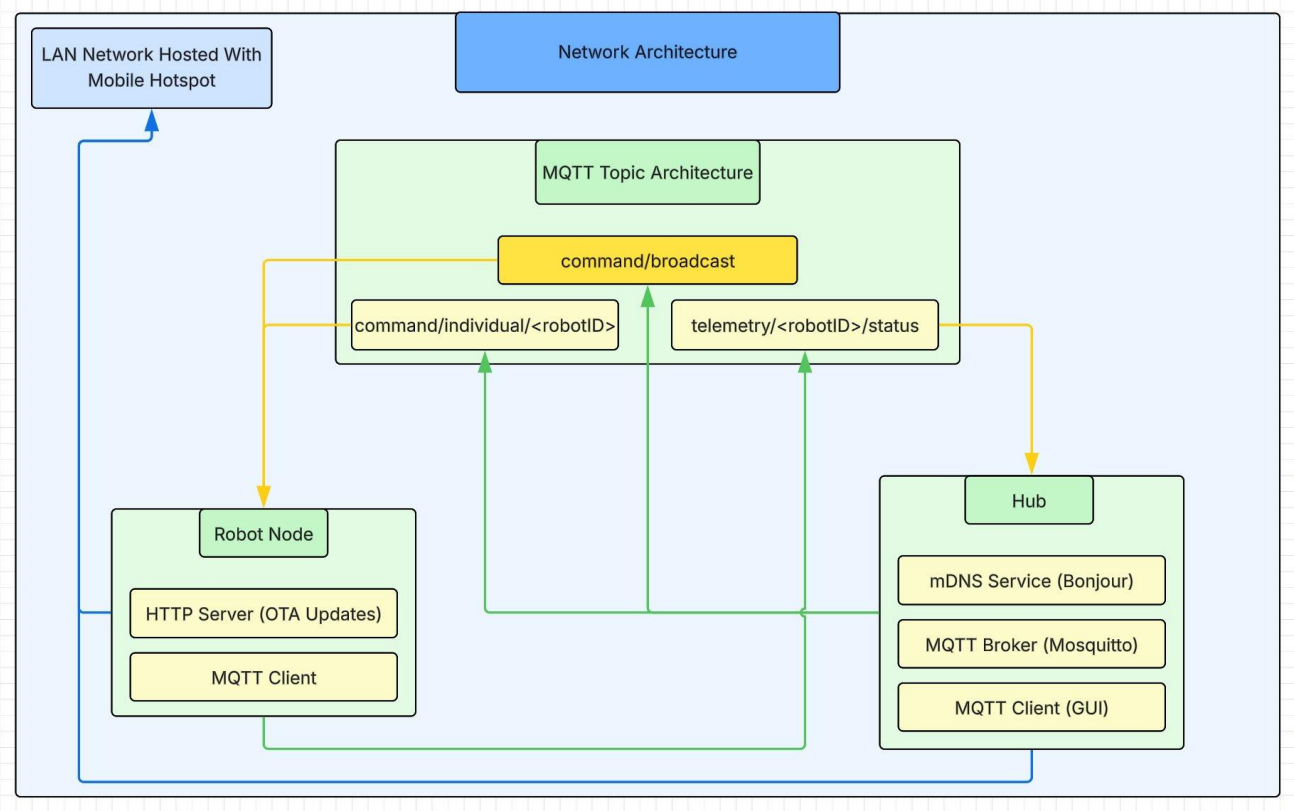
6.2.4.1. Network

The network architecture is based on a Local Area Network (LAN) established via a mobile hotspot. Each robot has the hotspot credentials hardcoded and automatically connects once the network is detected. Communication between the hub and the robots is handled using the **MQTT protocol**, with the **broker hosted on the hub**. Additionally, each robot runs a lightweight **HTTP server** that listens for **Over-The-Air (OTA)** update requests.

The hub hosts a **Graphical User Interface (GUI)** that allows the user to send commands to the robots through the MQTT broker and visualize real-time data streams from each unit.

In earlier iterations, hub-to-robot communication was implemented using HTTP, with each robot hosting its own server for command exchange. This approach proved inefficient, introducing unnecessary network overhead and significant latency. Migrating the communication layer to MQTT dramatically improved responsiveness, reduced network load, and provided a more reliable, event-driven communication structure.

Since the network is hosted on a mobile hotspot, the **DHCP server** on the phone (acting as a router) may assign different IP addresses to the hub each time the network is established. To ensure consistent connections, the hub runs an **mDNS (Bonjour) service** that broadcasts its hostname to the router upon connection. This allows the robots to connect via hostname rather than a fixed IP address, as the router dynamically resolves the hostname to a valid IP each time.



Picture #14 – Network Architecture Diagram

6.2.4.2. Individual Robot

Each robot's software operates as a **finite state machine**, governed by the values stored in the State structure. This structure contains:

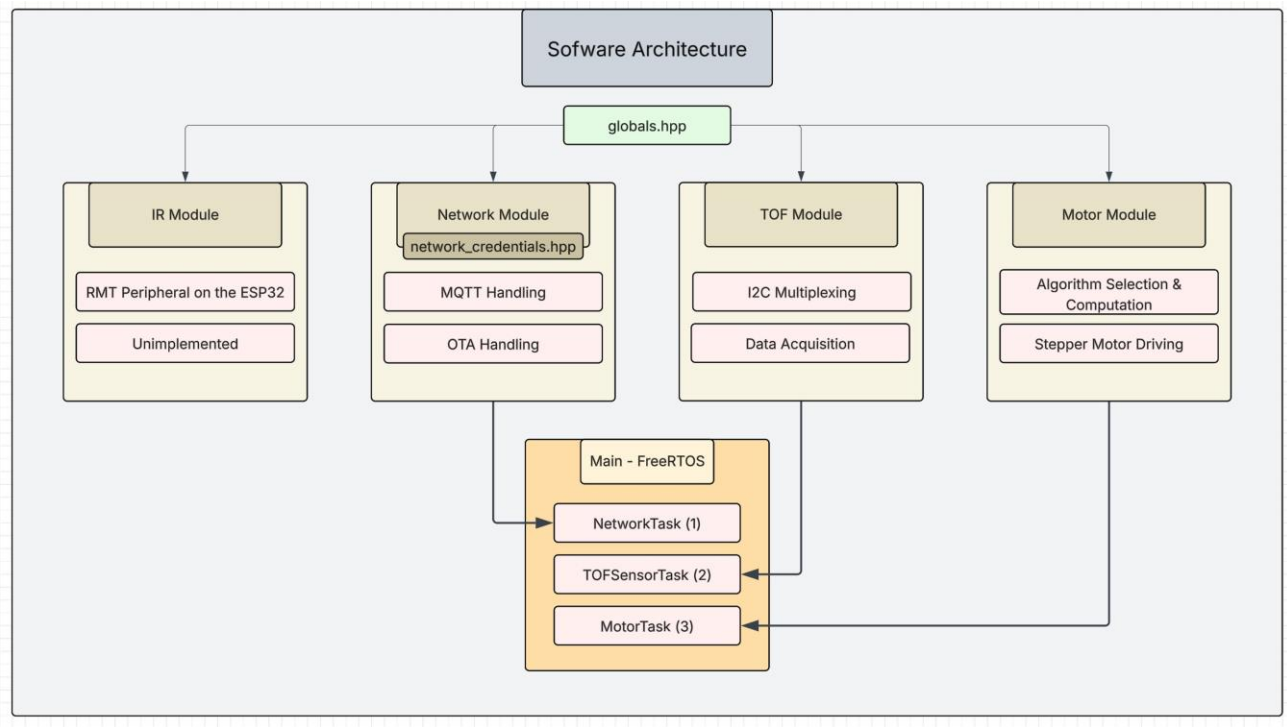
- An array of distance readings representing the distance to the nearest obstacle in each of six directions, continuously updated by the sensors.
- A mode enumeration that specifies the robot's current mode of operation.
- Seven additional variables used by one or more operational modes.

The system is built on **FreeRTOS** and consists of three concurrent threads:

- **Low-priority Network Thread (Core 0):** Handles the network stack, OTA updates, and incoming HTTP requests from the hub.
- **Medium-priority ToF Sensor Thread (Core 1):** Continuously reads sensors in a round-robin fashion and updates the State structure in a thread-safe manner.
- **High-priority Motor Control Thread (Core 1):** Sends pulse signals to the stepper motor drivers at a fixed interval of **1 ms**. This thread also determines the optimal movement

direction based on the robot's current operational mode and the sensor data stored in the State structure.

Additionally, there is a **fourth, currently unimplemented thread** intended for **infrared (IR) communication** between robots. This thread was planned to handle inter-robot data exchange using the onboard IR transceivers but was left unimplemented due to time constraints.



Picture #15 – Robot Software Architecture Diagram

In the following subsections, the implementation of the three main threads launched from the main() function is described in detail:

6.2.4.2.1 Network Module

The **Network Module** is responsible for establishing and maintaining communication between each robot and the hub. It exposes four primary functions in its header file:

- **ConnectToHub()** – Handles Wi-Fi connection and reconnection logic using the *WiFi* library.
- **SetupOTA()** – Initializes Over-The-Air (OTA) update functionality, managed via the *ArduinoOTA* library.
- **SetupServer()** – Configures the MQTT connection to the broker and subscribes to all relevant topics.

- **NetworkTask()** – A FreeRTOS task responsible for executing all network-related setup and maintenance operations.

The **MQTT setup** is designed as follows: each robot subscribes to two topics — command/broadcast and command/individual/<robotID>. The hub publishes broadcast commands to the first topic and targeted commands to the second. The network module includes an **MQTT callback function** that triggers whenever a new message is received. Upon receiving a message, the module checks whether it differs from the previous one, and if so, performs a **thread-safe update** of the global state structure.

Each robot periodically publishes its status to the topic telemetry/<robotID>/status. In future iterations, as more telemetry data becomes available, this structure can be expanded into multiple topics—for instance, to include metrics such as battery level, temperature, or signal strength.

All additional functions defined within the corresponding .cpp file act as **helper functions** that support and extend these core operations.

6.2.4.2.2 TOF Module

As previously mentioned, the six **Time-of-Flight (ToF)** sensors are connected through an **I²C multiplexer**. Consequently, this module is responsible for handling the **multiplexing**, **initialization**, and **operation** of the ToF sensors.

The .hpp file exposes two primary functions to main:

- **InitAllToFSensors()** – Performs initialization logic for the ToF system.
- **TOFSensorTask()** – The **FreeRTOS task** that handles continuous distance measurements.

The initialization function configures the ESP32's **I²C peripheral** and sets up the necessary variables and parameters required by the **VL53L0X** sensor library.

The **ToF Sensor Task** iterates through each sensor in sequence, using a **round-robin** style loop with a hardcoded interval of **5 ms** between readings (this value can be modified at compile time through a preprocessor #define). To ensure data integrity, the task uses a **mutex** to perform thread-safe updates to the shared State structure. This prevents race conditions that could occur if the **motor module** attempts to read from the State while the **sensor module** is writing to it.

6.2.4.2.3 Motor Module

The **Motor Module** is the most complex of the three software modules, as it is responsible not only for generating motion but also for making decisions about *where* to move based on the robot's current State variable. Consequently, this module contains several layers of logic, ranging from low-level motor control to high-level swarm behavior algorithms.

The **first layer** handles **stepper motor setup** using the **AccelStepper** library. Each motor is configured with a fixed maximum speed and acceleration and bound to its respective control pins on the microcontroller.

In retrospect (as discussed later in *Results & Comments*), dedicating more time to refining the motion control layer could have led to significantly smoother and more coordinated swarm behavior.

Building on this foundation, the **second layer** implements **basic motion control functions**, including:

- moveMotors()
- stopMotors()
- moveTowardsSensor(i)
- spinClockwise()
- setMotorSteps()

These functions abstract low-level motor commands and simplify the implementation of the higher-level decision-making algorithms.

The **third layer**, the **algorithm layer**, defines the robot's operational modes. This layer interprets the State structure and determines how the robot should behave in its environment. The following five modes are implemented:

- **OFF** – The robot remains stationary.
- **IDLE** – The robot maintains a minimum distance from surrounding obstacles in all directions.
- **MANUAL** – The robot can be manually controlled from the hub.
- **LINE** – The robots align into a single-file formation.
- **POLYGON** – The robots arrange themselves into a polygonal formation (in this case, a triangle).

In the following subsections, each algorithm used to achieve these behaviors is described in detail.

Each of the algorithms within the **algorithm layer** follows a three-part structure:

1. **Sensor Mask Generation:**
   A routine generates a **sensor mask** based on the specifications defined by the current mode in the State variable. For example, in the *IDLE* routine, the mask encodes the blocked directions based on the minimum allowable distance from obstacles in each direction.
2. **Decision Logic:**
   A routine evaluates the generated mask to **determine the optimal movement direction**. This process typically involves comparing distance readings, applying thresholds, and selecting the direction that best satisfies the mode's behavioral criteria.
3. **Motion Execution Wrapper:**
   A wrapper function calls the appropriate **movement function** (from the basic movement layer) to execute the chosen action. This design cleanly separates decision-making from actuation, improving code clarity and modularity.

Each routine is thread-safe, using a mutex to read from the state variable to protect it from trying to read while the sensor thread is trying to write.

Before moving on to the modes, a few words on the FreeRTOS task that handles the motors. Essentially all it does is a thread-safe read of the state variable, based on which it selects which mode handler to call to decide the direction of movement.

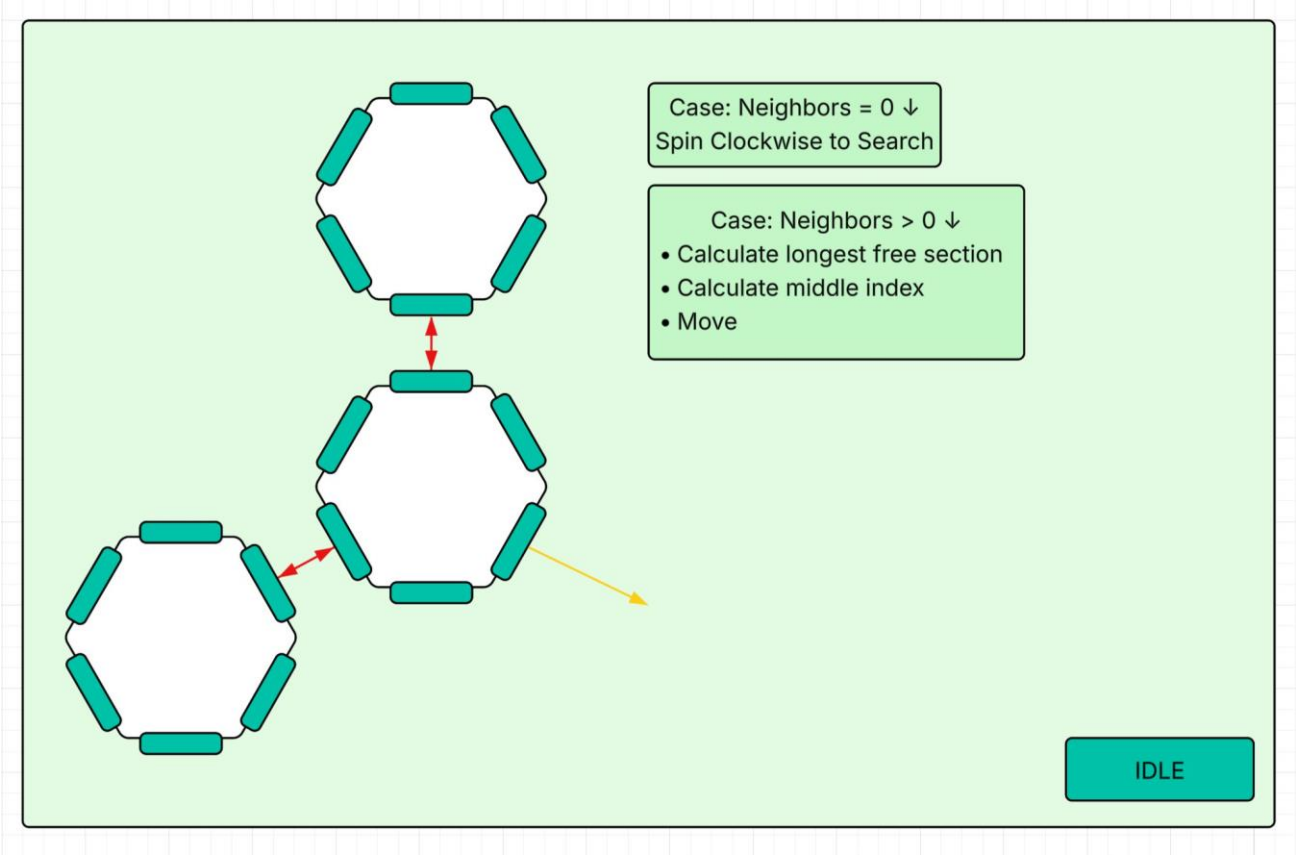
IDLE Mode

In **IDLE mode**, the robot attempts to maintain a safe distance from obstacles in all directions while remaining generally stationary unless adjustment is required.

A **6-bit sensor mask** is generated, where each bit corresponds to one of the six sensing directions. A bit value of **0** indicates that the direction is clear (*distance* > *Idle_thresh*), while a value of **1** indicates that an obstacle is too close.

The function **getBestMoveDirection_Idle()** processes this bitmask to identify the **longest uninterrupted free segment** and determines its central direction as the optimal movement heading.

The motion execution wrapper, **handleIdle()**, interprets the resulting decision: if the bitmask consists entirely of zeros (no nearby obstacles) or ones (completely surrounded), the robot remains stationary. Otherwise, it initiates movement toward the direction selected by the decision routine.



Picture #16 – Idle Mode Diagram

LINE Mode

In **LINE mode**, the robots arrange themselves in a **single-file formation**, maintaining approximately equal spacing between neighbors within a predefined tolerance. The desired inter-robot distance is specified by the hub and shared with each robot through the network.

The **sensor mask** in this mode evaluates the variable neighbors_maxDist, which defines the maximum distance within which an obstacle is considered a neighboring robot. Using this information, each robot identifies up to **two of the closest neighbors** detectable by its sensors.

Each robot (or node) can assume one of two roles:

- An **endpoint**, with only one neighbor.
- A **middle point**, with two neighbors detected on opposite sides.

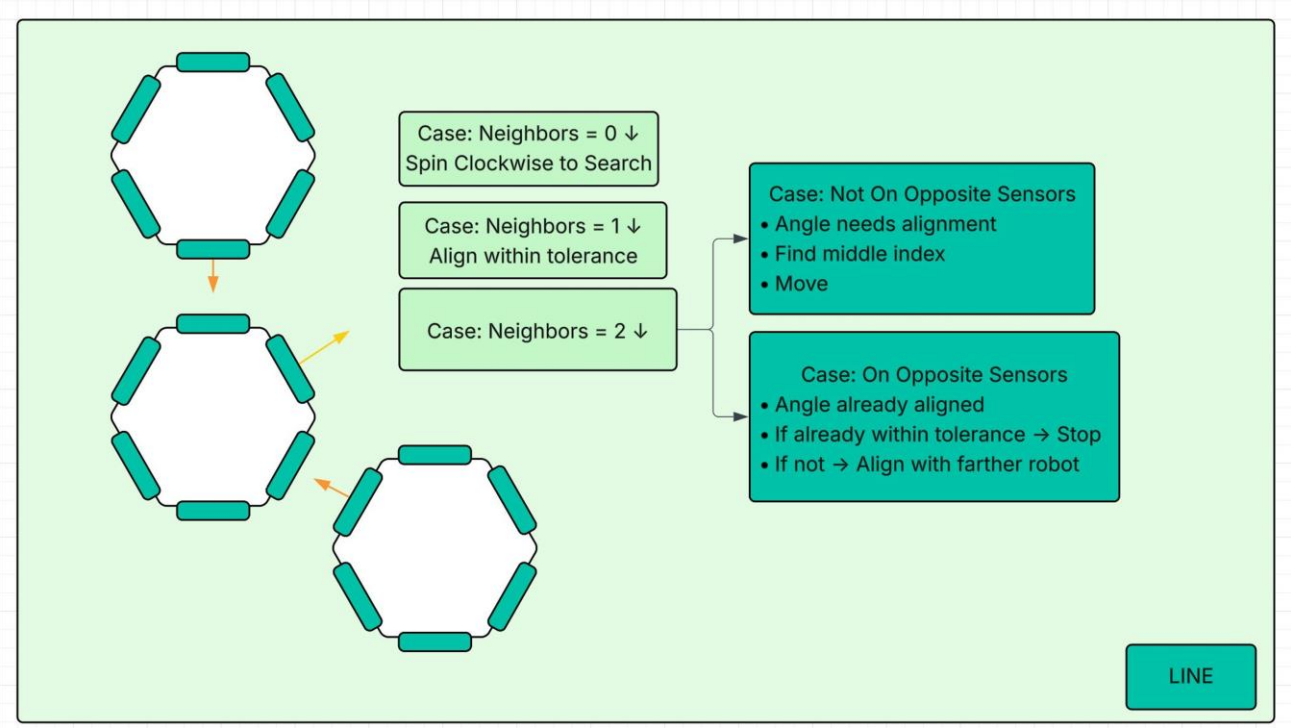
The algorithm distinguishes **three main cases**:

1. **No neighbors detected:**

The robot spins in place to search for nearby robots. A **wander routine** could be introduced here in future work to make this process more efficient and natural.

2. **One neighbor detected:**
   The robot attempts to align and maintain the desired spacing relative to this neighbor, using the line_alignTol and line_nodeDist variables as alignment and distance tolerance parameters.
3. **Two neighbors detected:**
   a. **Subcase A:** The two neighbors are positioned on opposite sides of the robot. In this case, the robot aligns itself with the **farther neighbor** to maintain consistent spacing across the line.
   b. **Subcase B:** The neighbors are **not on opposite sides**, indicating that the robot is misaligned relative to the rest of the line. To correct this, the robot moves toward the **midpoint** of the largest free region between the blocked sensors until it achieves proper alignment.

handleLine() simply executes the movement decided by **getBestMoveDirection_Line()**



Picture #17 – Line Mode Diagram

POLYGON Mode

The purpose of **POLYGON mode** is to arrange the robots into a **polygonal lattice** with a configurable number of sides (polygon_sides). Due to time constraints and the limited number of robots in the swarm, the current implementation focuses on forming a **triangle formation**.

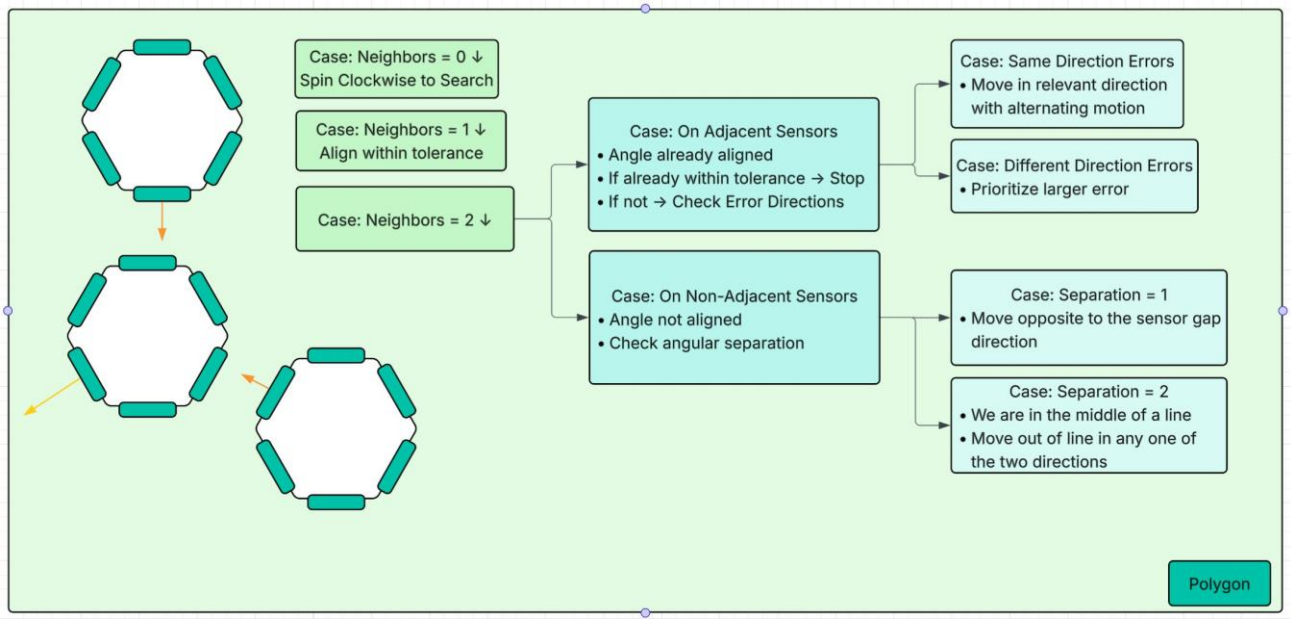
The possible lattice shapes are constrained by the **geometry of the robot**. Given the physical layout and sensing capabilities, the swarm can realistically achieve only **triangular or hexagonal lattices**. While more complex shapes could theoretically be formed, the current system focuses on the simplest feasible configuration given the hardware and swarm size.

The algorithm begins with a **sensor mask routine** similar to other modes, in which the robot selects the polygon_sides – 1 closest neighbors under the defined distance threshold.

The function **getBestMoveDirection_Polygon()** then evaluates the situation through a state machine:

1. **Zero neighbors:**
   The robot spins to search for neighbors. A **wander routine** could be implemented in the future to improve exploration efficiency.
2. **One neighbor:**
   The robot aligns with the detected neighbor within an acceptable margin, similar to **LINE mode**, and stops once alignment is achieved.
3. **Two neighbors:**
   Two subcases are considered:
   a. **Subcase A:** The two neighbors are aligned along **two adjacent sensors** of the target robot, forming the correct triangle configuration. Within this subcase:
      i. If the positional errors are mixed (one neighbor too far, one too close), the robot prioritizes movement toward the larger error.
      ii. If both errors are in the same direction, the robot moves toward the **midpoint between the two adjacent sensors**. Due to the limitations of the omni-wheel design, direct movement along this diagonal is not possible. To approximate the desired motion, the robot alternates movement: **0 cycles in one sensor direction, 50 cycles in the other**, effectively achieving an average motion along the diagonal.
   b. **Subcase B:** The two neighbors are **not correctly aligned**. Here, two situations are possible:

i. **Separation equals 1 sensor:** The robot moves in the **opposite direction of the gap sensor** to achieve proper alignment.
ii. **Separation equals 2 sensors:** The robot is effectively positioned in the middle of a line. To transition into the polygonal formation, it must leave the line from one of two directions. The algorithm selects one consistent arbitrary direction to break symmetry and achieve the triangular lattice.



Picture #18 – Polygon Mode Diagram

6.2.4.3. Hub

The hub is implemented on a laptop running a **Graphical User Interface (GUI)** that streamlines communication and coordination with the swarm robots over **MQTT**. The **MQTT broker** is hosted locally on the hub, making it the central point for command distribution and telemetry aggregation. At the same time, the hub also functions as an **MQTT client**: the GUI both publishes commands and subscribes to telemetry topics to receive real-time robot data.

The GUI provides several key functionalities:

- **Status Streaming:** Each robot publishes its status once per second to its designated telemetry topic. These updates are displayed in real time within the GUI. Although not fully scalable, the demonstration version includes dedicated status windows for each of the three robots.

- **Connection Indicators:** Colored indicators in the GUI show the connectivity state of each robot:
    - Connected
    - Stale (no message for 2–5 seconds)
    - Disconnected (no message for more than 5 seconds)

  Additionally, the GUI indicates whether the MQTT client itself is connected to the broker.

- **Manual Control:** Allows direct user control of any robot's movement.
- **State Update Commands:** Enables remote modification of all State variables of a robot.
- **Targeted and Broadcast Commands:** Supports both individual and broadcast communication via the corresponding MQTT topics.

This MQTT-based system replaces the previous HTTP-based communication scheme, where each robot hosted its own HTTP server. The new approach significantly reduces latency, minimizes network overhead, and enables **asynchronous, scalable communication** between the hub and the swarm.

Future enhancements could include extending the telemetry system to incorporate richer sensor data, allowing the GUI to visualize environmental information in real time—such as formation status, mapping data, or obstacle proximity.

6.3. Result & Comments

The resulting project serves as a **proof of concept**, demonstrating how small, simple robots with basic movement and communication capabilities—designed to be mass-produced at low cost—can be used collectively to create complex systems capable of addressing sophisticated problems.

As with my previous projects, the implementation proved more challenging than initially anticipated. The final outcome falls short of the original vision, which, at this stage in my experience, is expected for most projects and is not necessarily negative. For me, the **project-based learning process** involves setting ambitious goals: even if the final system does not fully achieve them, the experience provides valuable insights, technical growth, and a deeper understanding of both hardware and software design.

This section highlights **specific challenges** encountered during the project, including major concepts I had to learn, mistakes made, strategies that could have simplified development, and overall problems faced throughout the implementation.

PCB Design and Electronics

This project marked my first experience designing a **semi-serious PCB**. The initial five weeks were largely spent learning PCB design, consulting electrical engineering forums, seeking guidance from professors, and iterating through multiple **design–review cycles**. Ultimately, this effort resulted in a functioning PCB on the **first attempt**.

However, achieving this milestone required mastering **SMD soldering techniques** and identifying design improvements for future projects. For example, adding **testing probes**, **extra footprints for tweakable components**, and **SMD jumpers** would simplify debugging and modifications.

Once the first PCB was validated, the electronics portion proceeded relatively smoothly. In retrospect, it would have been more efficient to design the two PCBs to **stack rigidly** rather than rely on connectors and wires, as manually cutting and soldering all interconnections was time-consuming. Ideally, the **sensor modules** should have been designed as small PCBs that fit directly into their intended slots, which would have significantly reduced assembly time and complexity.

Networking

Another significant area of learning throughout this project was **networking**—specifically, enabling reliable communication between the robots and the hub, and selecting the most suitable protocols. I experimented with multiple network configurations, including using a dedicated plug-in router to create a LAN. In practice, however, a **mobile hotspot** served the same purpose while offering greater flexibility for deployment.

Initially, I implemented communication using the **HTTP protocol**, as it was the most familiar to me. This approach worked well for the three-robot demonstration but proved inefficient in practice. Each robot hosted its own HTTP server and processed incoming commands individually, which introduced latency and would not scale well for larger swarms where many robots need to receive the same command simultaneously.

To overcome these limitations, I later **migrated the network module to MQTT**, which greatly reduced latency and improved reliability. This change shifted much of the processing workload from the individual robots to the **hub**, where the MQTT broker is hosted, allowing for centralized message handling and asynchronous communication.

Another interesting challenge I encountered was the issue of **dynamic IP assignment**—since the hub's IP could change each time the hotspot reconnected. Setting up an **mDNS server** on the hub solved this problem by allowing the robots to connect using the hub's hostname instead of a fixed IP address.

Overall, this process deepened my understanding of networking principles and protocols such as **DHCP**, **DNS**, **HTTP**, and **MQTT**, as well as the interaction between different network layers in distributed embedded systems.

Software

One of the key aspects I focused on throughout the project was **modularity** in software design. Each module was written to be as self-contained as possible, allowing individual components to be modified or replaced with minimal impact on the rest of the system.

This approach proved especially valuable later in development, when I migrated the network module from HTTP to MQTT. Thanks to the modular structure, I only needed to modify a single file — network_module.cpp — and add a few lines to network_credentials.hpp. This flexibility significantly simplified the transition and demonstrated the importance of a well-structured, modular codebase.

This is a skill I'm still working on, but one I believe has gotten much better since my work on my Junior Project.

Movement and Algorithm Limitations

One of the most significant limitations of the current implementation is the **robot movement**. The motors exhibit jittery behavior, and the movements are not smooth. Furthermore, movement has been restricted to a limited set of predefined directions, which significantly constrains the robots' capabilities.

Ideally, more time would have been dedicated to refining the **movement stack**. Avoiding hardcoded acceleration values and implementing more robust and smooth **basic movement functions** would have allowed the higher-level algorithms to fully exploit the stepper motors, resulting in **more precise, fluid, and coordinated swarm behavior**.

Future improvements could include:

- **Movement smoothing:** Implementing moving average filters using the robot's movement history.
- **Advanced motor control:** Allowing the three motors to operate at different speeds and accelerations, enabling movement in directions other than the six sensor axes and supporting circular trajectories.
- **Synchronized motion:** Currently, robots do not consider sensor alignment when forming shapes. A coordinated alignment routine could allow robots to move in sync while maintaining proper formation.

Unimplemented Features

Two additional features remain unimplemented: **IR communication** and **ESP-NOW communication**. These are closely related, as the initial goal was to use the IR transceivers to relay each robot's unique ID to neighbors, allowing them to establish ESP-NOW connections. This could facilitate message passing across the swarm, for example, from one end of a line to the other.

Cost Considerations

The **Bill of Materials (BoM)** for each robot totals approximately **100 GEL**, which is prohibitively expensive for a single unit of a swarm. While mass production would reduce costs—through direct PCB implementation and eliminating 3D-printed parts—this issue warrants attention, as a key goal of the project was to demonstrate that a swarm solution can be **more cost-effective and efficient** than a single robot solving the same problem.

7. Conclusion

Working on this project has been an **invaluable experience**. I have refined many existing skills and acquired a range of new ones, including PCB design, electronics integration, networking, and swarm algorithm development.

The project also highlights the **challenges of designing and implementing swarm robotics systems**, from hardware constraints and motor control limitations to network scalability and software architecture. These challenges provided substantial learning opportunities, reinforcing the importance of iterative design, modular development, and realistic goal-setting.

While there is **considerable room for improvement**, the project demonstrates the potential of small, low-cost robots to collectively perform complex tasks, illustrating how swarm solutions can provide both **efficiency and scalability** beyond what a single robot can achieve. Future work could extend the system's capabilities, improve robustness, and explore new swarm behaviors, ensuring that the foundation laid by this project continues to yield valuable insights.

Overall, the experience has been both **technically enriching and personally rewarding**, confirming the value of hands-on, project-based learning in developing practical engineering and problem-solving skills.

Picture #19 – Three Robots Side by Side

8. List of References & Resources Used

https://www.youtube.com/watch?v=TYaquGrGhfk

https://www.youtube.com/watch?v=G94FDMGLwCc&t=14s

https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-devkitc-1/user_guide.html

https://www.alldatasheet.com/datasheet-pdf/pdf/551593/MPS/MP1584EN.html

https://www.mdpi.com/2504-446X/7/4/269

https://dspace.mit.edu/handle/1721.1/108816
https://github.com/ShapeLab/SwarmUI
https://github.com/yorkrobotlab/openkilo
https://link.springer.com/article/10.1007/s11721-019-00163-0
https://www.semanticscholar.org/paper/Swarm-Robotics-Pattern-Formation-Algorithms-Wagdy-Khalil/692f6c9e0dcaf77ed25d176130a857a6e070c85e
https://ieeexplore.ieee.org/document/8553906
https://www.researchgate.net/publication/221786517_Formations_of_Robotic_Swarm_An_Artificial_Force_Based_Approach
https://arxiv.org/abs/2309.01240
https://www.youtube.com/watch?v=fDrIpZ390pE
https://www.youtube.com/watch?v=t8D64Lbh2YY&list=WL&index=20
https://www.youtube.com/watch?v=_e1Jwueaxhs&list=LL&index=2
https://www.hivemq.com/static/ebooks/hivemq-ebook-mqtt-essentials.pdf

9. მადლობის გვერდი

მინდა მადლობა გადავუხადო შემდეგ ადამიანებს:

- **ზვიად სულაბერიძეს** - რიგ საკითხებში დახმარების გაწევისთვის.
- **შალვა ლაღაძეს -** SMD პაიკაში გაწეული დახმარებისთვის
- **ირაკლი დონგუზაშვილს** - ლაბორატორიაში არსებული ინვენტარის 90%-ის წყაროობისთვის.
- **ალექსი ახალაიას** - PCB დიზაინში გაწეული დახმარებისთვის
- **ავთო მღებრიშვილს** - პროექტის ბოლოსკენ მისი პრინტერის თხოვნისთვის.
- **გუგა ვარდიაშვილს** - ჩვენი კურსის გვერდში დგომისთვის.
- **მთლიან ECE21-ს** - იმისთვის, რომ საუკეთესოები არიან.

Appendix

https://github.com/AlwaysAbia/senior-project-freeuni