Multi-Unit Reconfigurable Robot

Abstract

As an innovative type of robotic system, reconfigurable robots have the ability to change their physical configuration or shape to adapt to various environmental requirements. Compared with their traditional counterparts, these robots can perform tasks in unpredictable, constantly evolving, and dangerous environments more effectively, e.g., conducting search and rescue operations in disaster zones or exploring uncharted territories. Previous or ongoing research in reconfiguration robots has primarily focused on improving modularity, flexibility, and efficacy. In this project, though, we aim to design a novel modular construction for reconfigurable robots, which depend on specifically designed joints to allow the robot to accommodate even more types of surfaces in extreme environments. Our system consists of multiple units, each having three joints and providing great versatility to alleviate energy loss. By mimicking animal joints, our joints can perform more movements. Our research has shown that our innovatively designed and integrated hardware and software with modular construction exhibits the ability to move in a single unit, two units, or four units to adapt to a greater variety of challenging tasks. A single unit can move at a speed of around seven centimeters in slightly less than two seconds, and when four units are combined, it can move at a speed of around fourteen centimeters in three seconds. Besides, this system of multi-unit and reconfigurability can be expanded to create robots that are capable of different tasks, such as robotic arms. Also, when combined with other current technologies, it can even be used more widely.

Keywords: Reconfigurable, Multi-Unit, Multiunit Software Control, Modular Robotics

Table of Contents

Multi-Unit Reconfigurable Robot 1
Abstract 2
Table of Contents 3
1 Introduction 6
1.1 Background 6
1.2 Current State of the Study 6
2 Design Plan 9
2.1 Modularity 9
2.2 Options of Multi-unit Connections and Sided Connections 9
2.3 Programming and Hardware Choices 9
3 Mechanical Structure Design 10
3.1 Overall Structure 10
3.2 Unit Design 10
3.2.1 Material Choice 10
3.2.2 Subunit/Joint Design 10
3.2.3 Servo Mounting Mechanism 11
3.2.4 Subunit Connections 13
3.3 Connector Motor Mounting Mechanism 14
3.4 Connector and Receptor Design 14
4 Hardware/ Electronic Design 17
4.1 Hardware Choices 17
4.1.1 Mainboard 17
4.1.2 Servo 18
4.1.3 Motor 18
4.1.4 Battery 19
4.2 PCB Design 19
4.2.1 Power Supply/Management Unit 19
4.2.2 Motor Control Unit 19
4.2.3 Servo Control Unit 20
4.2.4 Mainboard Connection Unit 20
4.2.5 Final PCB 21

5 Software Design/ Programming 24
5.1 Overview - Arduino 24
5.2 Wireless Communications 24
5.2.1 Unit-to-Unit Communications – V1 24
5.2.2 Unit-to-Unit Communications – V2 25
5.2.3 Device-to-Unit Communications – V1 25
5.2.4 Device-to-Unit Communications – V2 26
5.2.5 Device-to-Multi-Units Communications 27
5.3 Multitask Handling 29
5.3.1 Benefits of Multitask Handling 29
5.3.2 Achieving Simultaneous Execution 29
5.4 Movement Control 29
5.4.1 From-Device Movement Control V1 & V2 29
5.4.2 Data Packaging – V1 30
5.4.3 Data Packaging – V2 31
5.4.4 Motor Control – V1 31
5.4.5 Motor Control – V2 31
5.5 Unit Movements 32
5.5.1 Overview 32
5.5.2 Motion Control Program/ Process 32
6 Experiments of Final Version 33
6.1 Single Unit Test 33
6.2 Two Units Test 36
6.2.1 Movement Calculations 36
6.2.2 Experimentation 37
6.3 Four Units Tests 39
6.3.1 Four Units Turn Tests 39
6.3.2 Four Units Forward Tests 40
6.4 Servo Strength Test 41
6.4.1 Overview 41
6.4.2 Servo Strength Results 41
6.4.3 Analysis 42
7 Conclusion 43
7.1 Future Outlook 43

8 References 44

1 Introduction

1.1 Background

As the demand for robots continues to expand and application scenarios become increasingly complex and diverse, the adaptability requirements for robots are also increasing. These special needs have given birth to modular robots, which have become an important branch of robot technology.

Modular reconfigurable robots are composed of a group of individual units through various ways of connectivity. Rigid connections are established between the modules mechanically or magnetically, forming tight connections. Through the contraction, extension, rotation, and other motion modes of joints within each unit, the entire robot can achieve more flexible mobility and adapt to complex environments. In this structure, the coupling between modules and the design of the joints is crucial for the system's overall functionality. The modules can have various shapes, such as square, polyhedral, sheet-like, cylindrical, etc. Through combination, different motion effects can be achieved. [1]

In these systems of units, each module has more flexible mobility. The greater the degrees of freedom of a module, the more flexible its motion will be. However, these systems' mechanical design and motion control become substantially more complex.

Besides simple mobility, modules can be selectively made (provided they have proper connection methods) for the robot, such as object holding, transportation wheels, photography cameras, distance and color sensing, environmental data testing, etc. Under such a modular framework, modules can be freely assembled to meet various requirements or be modified on the fly.

1.2 Current State of the Study

One of the studies looked at the connection between units for robot arms in an industrial setting. It includes complicated PCB designs and sturdy locking mechanisms to guarantee a tight connection between robot arms. In a research paper published on October 28, 2020, called "Intelligent Modularized Reconfigurable Mechanisms for Robots: Development and Experiment," authors Wenfu Xu, Liang Han, Xin Wang, and Bin Liang argue that traditional industrial manipulators with a single configuration are difficult to meet a variety of tasks now required in factories. Therefore, they investigated the types of joints that could be used in reconfigurable robot arms. They compared two lightweight connection mechanisms with each other. [2]

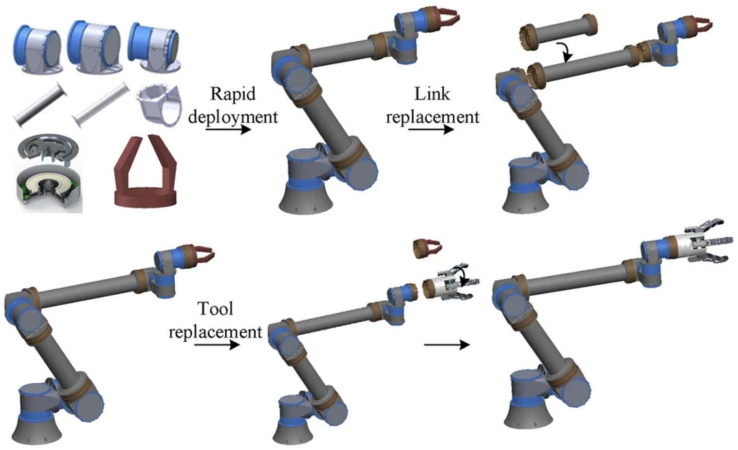

Figure 1 - Reconfigurable Robot Arms

On the other hand, these types of joints are not necessarily used in moving parts or robots. Some researchers apply this modular idea with sturdy joints and apply it to stationary, non-moving objects, such as buildings. Such as the research called “Mechanical Behaviors of Inter-Module Connections and Assembled Joints in Modular Steel Buildings: A Comprehensive Review.” This study creates an overview of the methods for creating inter-module connections. This modular approach to building allows a more time-saving approach to building, reduces on-site work, and wastes fewer resources during such processes. [3]

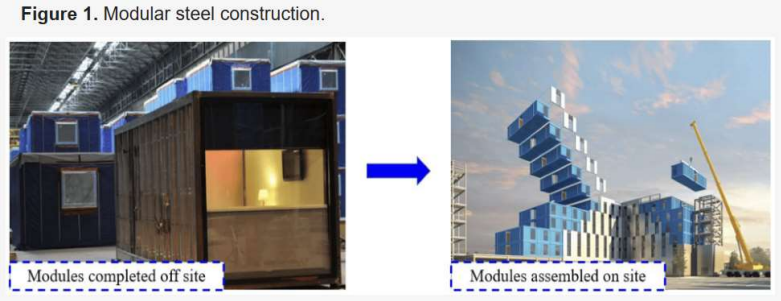

Figure 2 - Joints in Modular Steel Buildings

SWARM-bots were researched and designed by Dr. Marco Dorigo of the Free University of Brussels in Belgium. The SWARM-bot system adopts a track mode and communicates through communication modules, enabling robots to aggregate, work collaboratively, allocate tasks, self-assemble, and navigate. SWARM-bots robots dock using grippers. [4, 5] SWARM-bots can accomplish tasks that individual s-bot robots cannot. The following figure shows multiple s-bot robots in a series completing tasks such as crossing a ravine and climbing stairs.

Figure 3 – Swarm-bots

Figure 4 - Small-Sized Modular Robot

Another study uses magnetic connections instead of mechanical joints. This increases the tolerance when two units connect but only compensates for a weaker connection. In the Smores-Ep module, researchers used magnets to align units together and enabled the joint itself to rotate. [6]

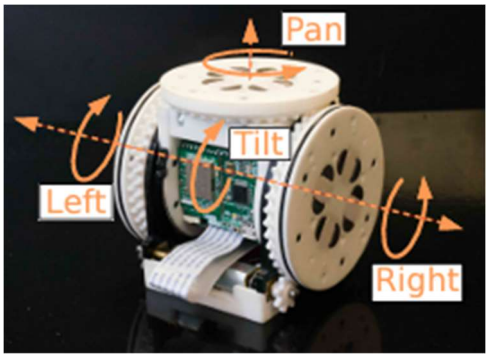

Figure 5 - SMORES-EP Module with Rotatable Connectors

While some studies dive deep into different joint styles and how to stabilize connections between units, other researchers also investigated algorithms to facilitate multi-unit control when they form into a single unit. For example, “Scalable multi-radio communication in modular robots” by V. Kuo and R. Fitch presented a multi-radio architecture for communication in modular robots capable of scaling and constant bandwidth neighbor-to-neighbor communication. In the research, the researchers used wireless RF links as the primary source of connection and validated the approach using a 45-radio testbed with real data loads. [7]

Another important and distinguishing characteristic of modular robots is that they are often capable of self-reconfiguration based on the environment and task (meaning they can reconfigure their arrangement without human or external assistance or interference). Often, many such rearrangement sequences will all result in the desired outcome, but the best method is always the one with the least amount of attachment and detachment. This is because the fewer actions, the less likely something will malfunction. In an article called “Auto-Optimizing Connection Planning Method for Chain-Type Modular Self-Reconfiguration Robots,” the authors, H. Luo, and T. L. Lam, proposed an algorithm that can auto-optimize connection plannings for multiple in-degree single out-degree modules. The solution contains a polynomial-time algorithm to calculate near-optimal solutions and an exponential-time algorithm to further optimize the solutions automatically when some CPUs are idle. [8]

2 Design Plan

2.1 Modularity

The aim of this project is to create a robot that can traverse all sorts of terrain with great flexibility. Section 1.2 shows that if we make the robot modular, it will have this flexibility. The robot will also need to move itself to some extent, and there are two options. The first is to create connectors that can move and rotate. The Second option is to develop non-moving connectors and make each module itself movable with joints.

The first option is easily viable with electro-magnet joints. Although such a joint creates relatively strong connections and great versatility, it consumes a lot of power while generating a lot of additional heat.

Although such a connection is also achievable through complicated mechanical structures, such structures would provide many points of weakness that, without expansive and high-strength materials, would be easy to break.

On the other hand, the second option traded off some movement margins to gain much stronger mechanical joints that are less likely to break or fail. I ultimately decided to choose this option over the previous one.

2.2 Options of Multi-unit Connections and Sided Connections

Since I decided to create a modular robot, we need to consider how they can connect with each other. The first type of connection required is one that forms a snake-like shape. This allows the robot to crawl forward and squeeze into tiny cracks. Besides, since the robot needs to be able to stand up and traverse, it is necessary for different modules to connect to each other from the sides. This connection allows the robot to stand up from the supports of all four sides.

2.3 Programming and Hardware Choices

In this project, the program has to be uncomplicated and quickly deployable while also having wide compatibility for all sorts of motors, servos, and sensors and being capable of remote communication. Therefore, I decided to go with Arduino. Arduino is an open-source electronics platform that combines both hardware and software. It revolves around Arduino boards, which are physical, programmable circuit boards (also often referred to as microcontrollers). These boards can read inputs from the environment—such as light detected by a sensor, a button press, or remote communications—and translate these inputs into outputs. Outputs might include activating a motor, turning on an LED, or sending a message through special channels. The Arduino Software (IDE) serves as an Integrated Development Environment for writing and uploading computer code to the physical Arduino board. The programming language is similar to C++ and Java; therefore, it is relatively simple to learn and start (although I already have a lot of experience with it). All of these features combined made Arduino the perfect option for this project.

3 Mechanical Structure Design

3.1 Overall Structure

As previously mentioned in section 2.1, the robot needs to be modular. Therefore, there must be units that can connect to each other and be able to move and change their shape. Also, each unit must be able to move itself. Therefore, I have decided to use 3 joints for each unit, 2 of which are in the same direction. This allows each unit to crawl forward pretty quickly. There are also connectors (one male and one female) on both ends of the robot. However, due to the type of connection I mentioned in section 2.1, two sides of the connector will be different: one male side and one female side.

The entire structure will be 3D printed with the single exception of the plate holding the motor for the connector, which is laser printed because it is mostly flat and is unnecessary to be 3D printed.

Besides, I also made different versions of different parts of the mechanical structure, as I think those could be improved. For example, I improved the subunit junctions, and how they would be more securely connected to each other through the rotational joints. These versions will be described in each section, which represents the specific part of the robot.

3.2 Unit Design

3.2.1 Material Choice

Since this project requires a rigid, 3D-printed chassis, the material must be 3D printable, sturdy, and bend-resistant. With the budget of this project in mind, the best materials that satisfy this need are PETG plastic and PLA plastic. PLA is known for its strength while being beginner-friendly and inexpensive, while PETG is known for its superior mechanical properties and impact resistance.

In the first two units of the robot, I used PLA plastic as it is the most frequently used plastic in 3D printing. It turned out well, but the overall structure is still bendable. Therefore, I printed the next two units with PETG plastic. Despite the support being more challenging to take off, the overall structure of the units is more stable and resistant to bending and warping. I ended up not changing the over PLA for PETG because when I compared them side to side, it seemed that the direction of the print mattered more than my material choice, so I simply reoriented my 3D models in the printing software and reprinted them.

3.2.2 Subunit/Joint Design

Each unit is ultimately made out of 3 similar subunits, each containing one 270-degree servo, a joint, and various connection types depending on whether it is a subunit on the side or in the center. This design would give each unit 3 degrees of freedom and the ability to move. Each subunit is then made from 2 U-shaped pieces. This design guarantees the integrity of the overall structure and the functionality of each subunit.

In the first functional prototype, I used a joint design with a plate that connects directly to the servo, which provides the power for rotation. However, in practice, I observed that this piece would actually tilt sideways because two joints are only connected via one link (and a not very stable one at best). Therefore, in the second version, I decided to add another link on the other side of the joint. This linkage will help support the connection and create more table joints.

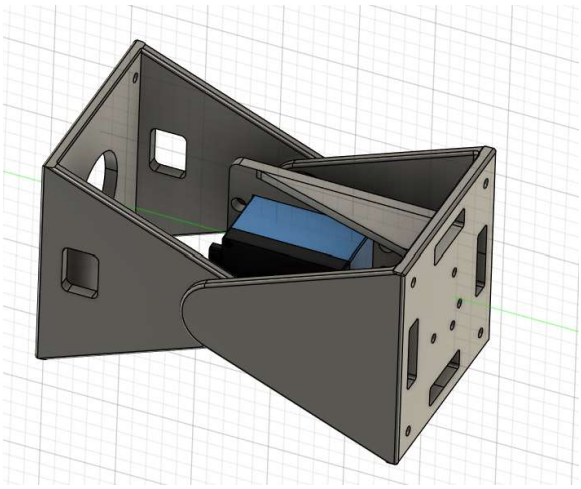

Figure 6 - Joint V1

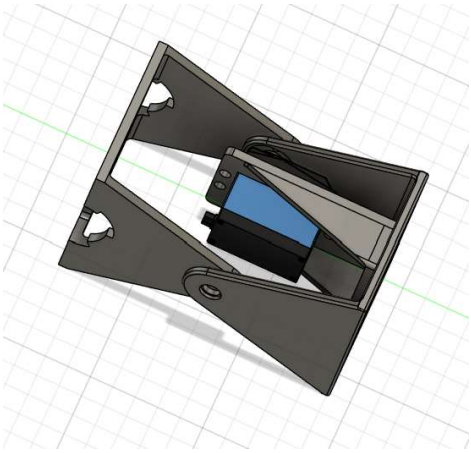

Figure 7 - Joint V2

3.2.3 Servo Mounting Mechanism

One of the three servos is mounted directly to one of the U-shaped plates of the subunit with 4 screws that penetrate through the subunit and extends into the other subunit. This method of connection additionally acts as a connection between subunits.

Because of the design and each subunit need to connect to each other, two of the servos share the same screw that penetrates 2 U-shaped plates of different subunits. This also acts as a connection between subunits.

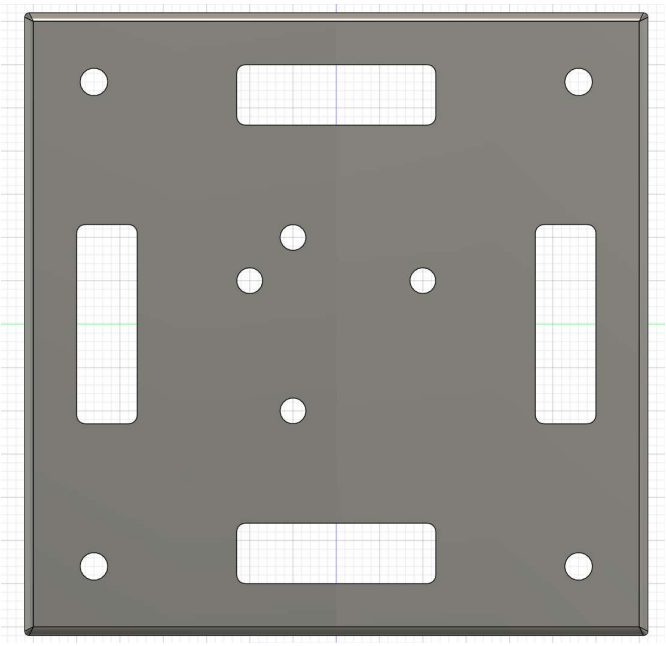

Figure 8 - Servo Mounting Base

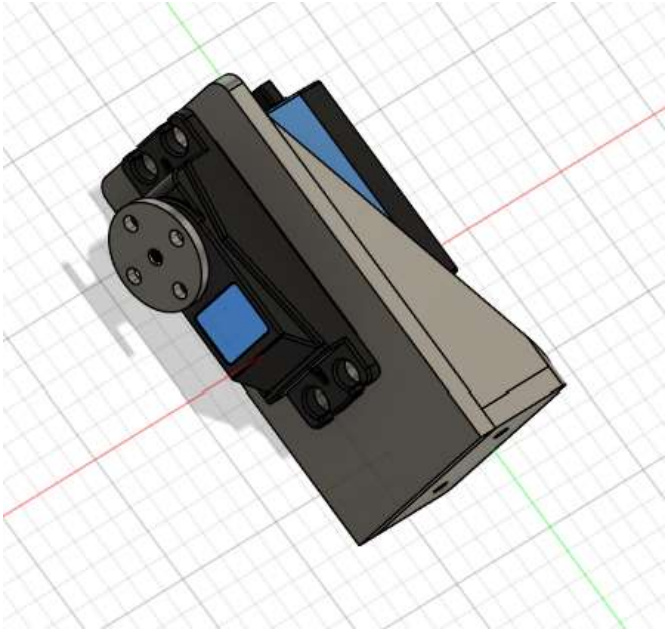

Figure 9 - Servo Mounting Plate

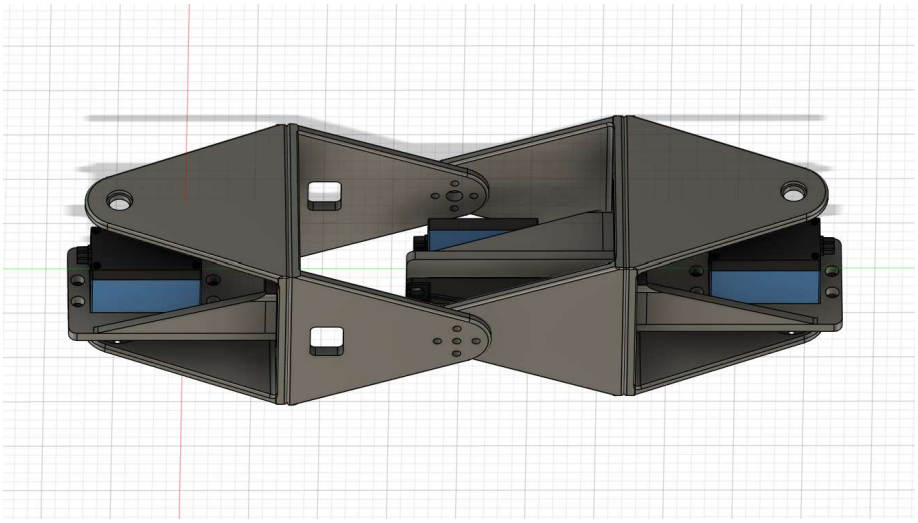

Figure 10 - Servo Mounting Positions

However, when the product is built, I cannot fully use all 4 screws because it is extremely difficult to squeeze long screws through the small opening it has and more difficult to tighten them with screwdrivers.

3.2.4 Subunit Connections

All subunits within each unit are connected to their neighbor/ neighbors with at least 4 screws on the connection side and an additional connection in the middle through the motor mounting positions (as seen in section 3.2.3). These screws guarantee the tightness of the connections and lock rotations on the connection plane.

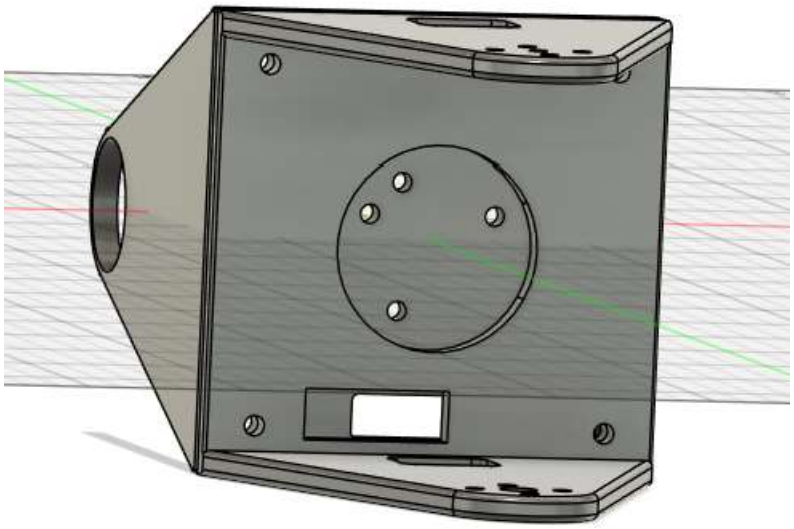

Figure 11 - Subunit Connections

3.3 Connector Motor Mounting Mechanism

This is only applicable to the male connector side of each unit. Each unit will have one N20 motor mounted to one end. The N20 motor is secured on a laser-printed plywood piece with a C-bracket, which is then secured to the robot through 4 standoffs at the corner of the plywood piece and the robot. The screws stick outwards of the end of the robot and act as anchors to lock the rotation of the two connecting pieces on the connection plane.

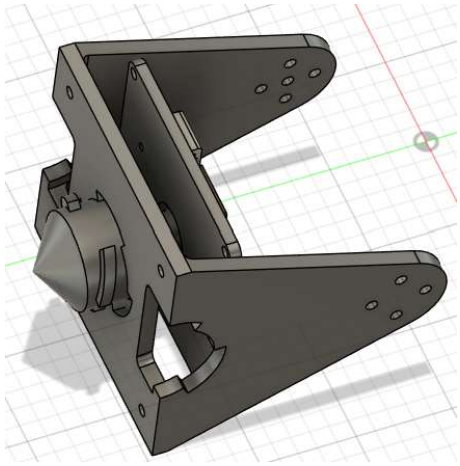

Figure 12 - Motor Mounting Plate

3.4 Connector and Receptor Design

This only applies to the two subunits at both ends of each unit. For the receptor end, there are four enlarged screw holes at the four corners of the face and a large opening at the center for successful connection. The four enlarged screw holes are designed to be just bigger than the size of the screws on the connector side. They stop the two connected units from rotating on the plane parallel to the two faces.

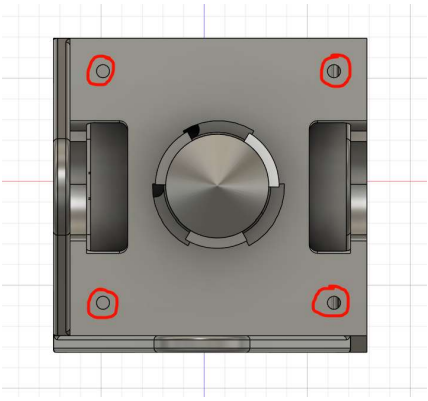

Figure 13 - Connector face with screw holes circled in red

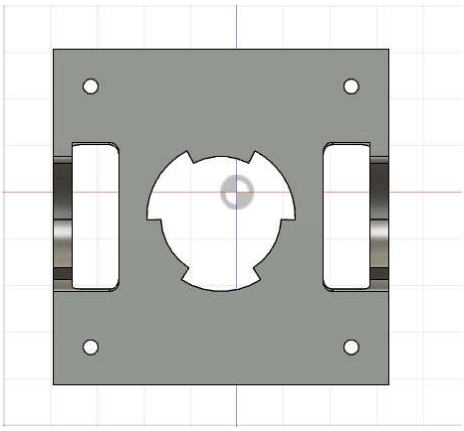

Figure 14 - Connection face

In this first version, the units can only be connected in a chain, not yet having the capabilities to connect sideways and allow for wider arrangement capabilities since there are no connectors on the side of the unit.

Then, in the second version, I created a large center opening on the sides of where the subunits initially connect and used a soldering iron to melt part of the sides off the subunit to act as screw holes. This allows additional connections on the sides of the original connection. This allows the robot to become a complete four-legged robot, which offers much more freedom of movement and greater mobility than just simply connecting it to form a line. Besides, despite the fact that part of the unit was cut off, it did not impair much of its structural integrity.

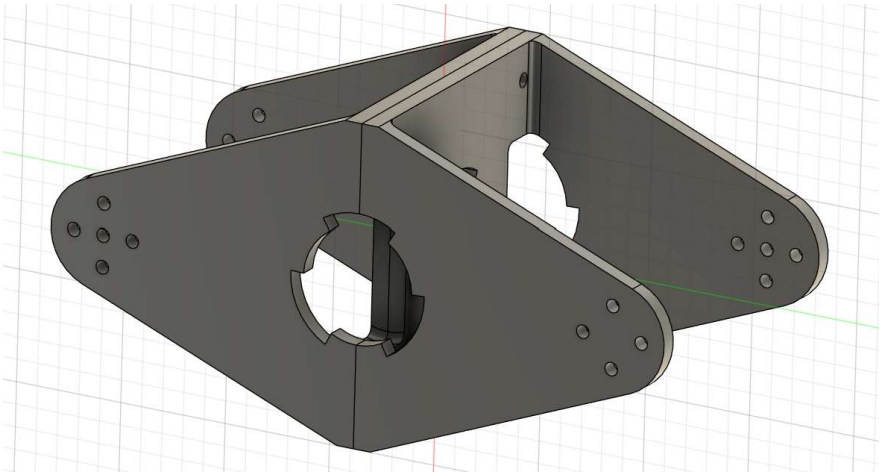

Figure 15 - Side connections formed by two units

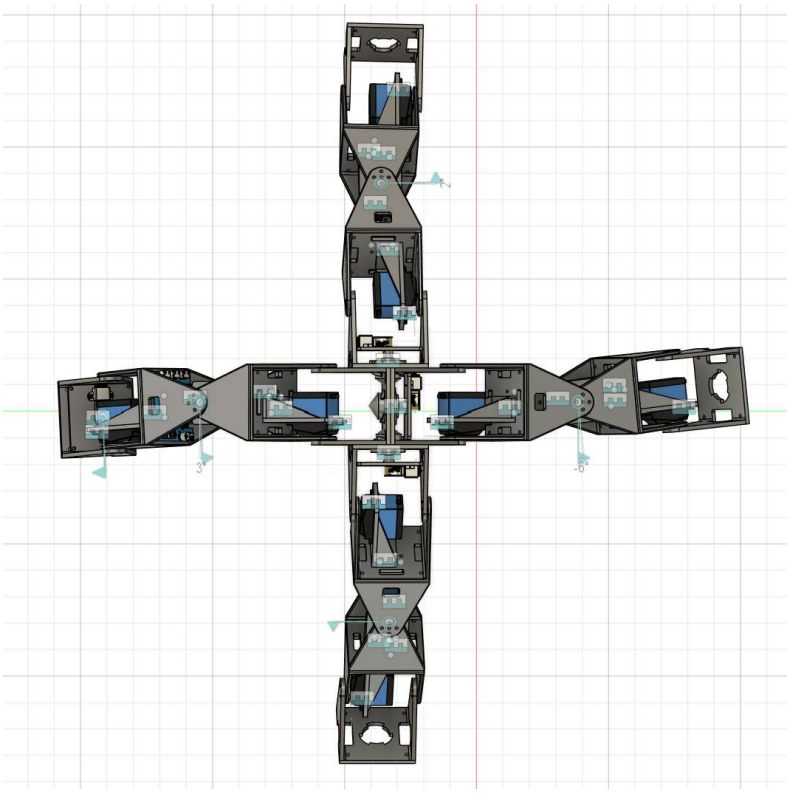

Figure 16 - Possible configuration with side connectors

4 Hardware/ Electronic Design

4.1 Hardware Choices

4.1.1 Mainboard

For this project, the mainboard must be powerful enough for fast communication through WIFI and BLE, with enough pins for 3 servos (each requires a 5V, a GND, and a GPIO pin) and 1 motor (it requires 2 GPIO pins). It also needs to be small enough to be placed nicely into a PCB smaller than 65mm * 60mm with many other components because of space limitations. Given such requirements, one option I considered is the Arduino Nano board. Despite being small enough to be placed into the PCB and have the power to compute that data, it lacks WIFI and Ble, which then requires additional pieces on the PCB that have no room.

Figure 17 - Arduino Nano

Another option is ESP32. It is also small enough to be placed into the PCB while also having WIFI and BLE. However, it does not have enough flash for the program that I need to upload with both WIFI and BLE.

Figure 18 - Regular ESP32

Then, I finally landed on a variant of ESP32: ESP32-S3. It has a bigger flash and bigger RAM, which is enough for me to upload my program while having every other feature that I need for this project. It is indeed the best fit for this project.

Figure 19 - Luatos ESP32-S3

4.1.2 Servo

The servo needs to have enough strength to lift the entire robot with 4 units in the future. The further away the center of mass is from the servo, and the heavier the robot gets, the harder it would be for the servo to lift the robot. We can approximate the force required for each servo such that they are enough to raise the end of the unit (with 1 other unit attached).

For the purpose of this project, I chose a servo that has a torque of 25, which is ideally enough for lifting the entire equipment.

Figure 20 - Dsservo digital servo 25KG

4.1.3 Motor

The motor is used to spin the connector piece to connect two subunits, as stated in section 3.4. It needs to be strong enough to spin the connector piece with a lot of friction and tighten it so that the two pieces would not be loosely connected. For the purpose of this project, I chose an N20 motor with a high torque ratio. The N20 motor uses 12V power and spins at a speed of 1 rotation every 2 seconds. This allows the motor to spin reliably from the current and power provided by the power supply.

Figure 21 - N20 motor with a torque gear ratio

4.1.4 Battery

The Battery mainly needs to satisfy two requirements: rechargeable and small enough to fit into the unit. In the end, I chose a battery with a somewhat small capacity, but it satisfies both of the requirements I set.

Figure 22 - 12V, 1200mAH Battery

4.2 PCB Design

4.2.1 Power Supply/Management Unit

This is the part of the PCB where the battery is connected to. It helps supply 12V power to the motor while providing 5V power to the mainboard through L7805 and servos through another 5V down-volt chip. This unit helps guarantee every unit on the PCB is sufficiently powered and can run regularly.

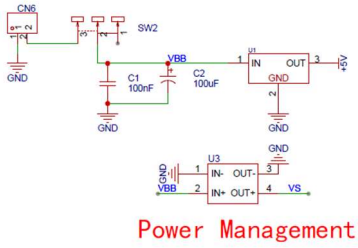

Figure 23 - Power Management/Supply Schematic

4.2.2 Motor Control Unit

Since the motor needs 12V power and two PWM inputs from the mainboard (ESP32-S3), it requires a special unit to control it. This unit provides power for the motor directly from the battery while also receiving information on which direction to spin from the mainboard.

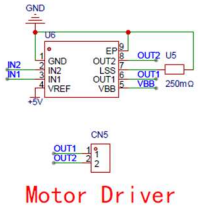

Figure 24 - Motor Control Schematic

4.2.3 Servo Control Unit

The servos are controlled with 5V power and a single PWM input from the mainboard. Therefore, a special unit is required to convert the 12V power from the battery to 5V, which the servo can use. The unit also takes input from the mainboard and allows the connected servo to turn to the input degree.

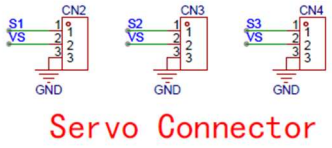

Figure 25 - Servo Control Schematic

4.2.4 Mainboard Connection Unit

This section did not exist in the first version of the mainboard because I did not think I would need other expansions for other components. However, I later thought about further expendabilities for my project, so I added these connection units. This allows expansions like cameras, etc.

This will mount the mainboard to the PCB and control everything. Besides all the pins used by the units mentioned above, there are also leftover pins. I then extended these pins to a centralized location for future use and additional hardware connections. I also added an indication light on whether if the mainboard have power or not (it does not necessarily mean that the Servos and Motor are powered).

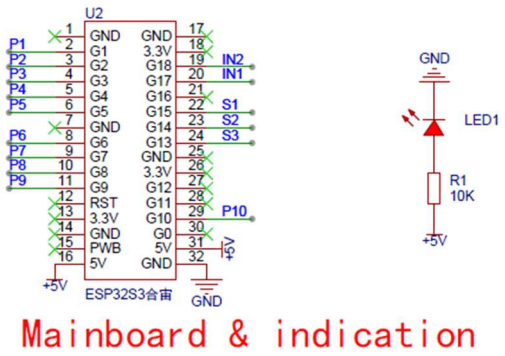

Figure 26 - Leftover Pins Schematic

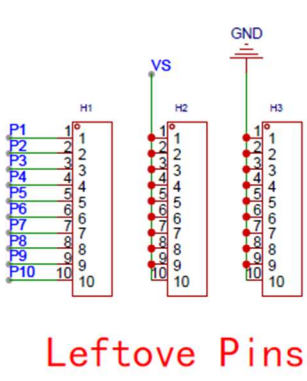

Figure 27 - Mainboard and Indication Light Schematic

4.2.5 Final PCB

This is the overall PCB schematic and the final product after finalizing the layout of the PCB.

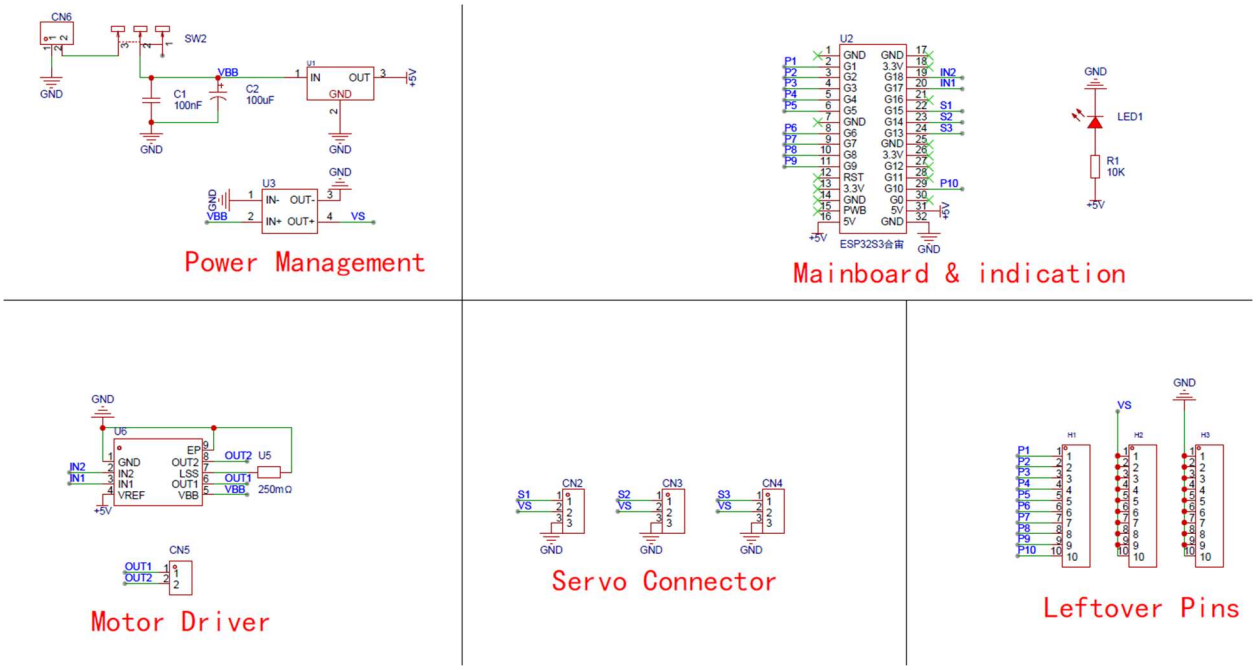

Figure 28 - Overall PCB Schematic

This is what the printed-out PCB looks like without any components attached. It only includes the raw circuits and labels left on the PCB. The arrows on the PCB indicate which way the component should be mounted, while some, such as a resistor, can be mounted in both directions.

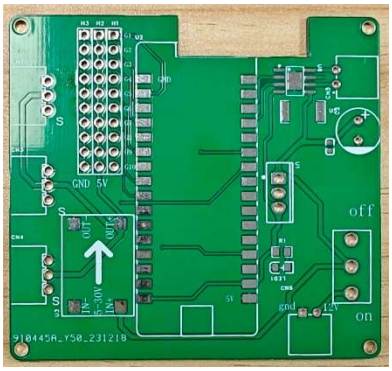

Figure 29 - PCB Product Without Components

The process of attaching various components is complicated. The first step is to solder on every surface-mounted device (SMD). This process requires some soldering paste (essentially powered solder suspended in flux paste) and a heating table. I first applied a moderate amount of solder to every place that needed to be soldered on the PCB and gently placed all the components on top of the paste without pushing them too hard. Then I placed it on top of the heating table, which heated up to about 230 degrees Celsius. I waited for a while and corrected every piece that needed realignment. After about 4 minutes, I took the PCB off of the heating table while making sure it would not tilt.

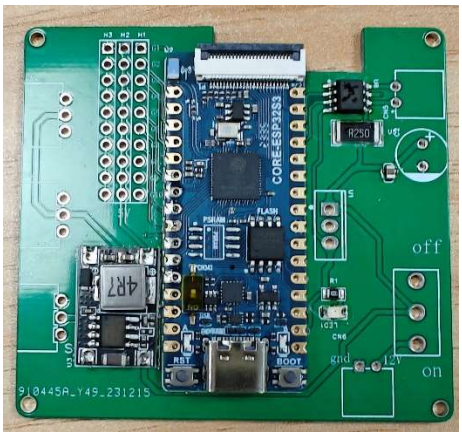

Figure 30 - PCB With Every Surface Mounted Component

After letting it cool for about 5 minutes, I soldered every through-hole component (THC) to their respective locations on the PCB.

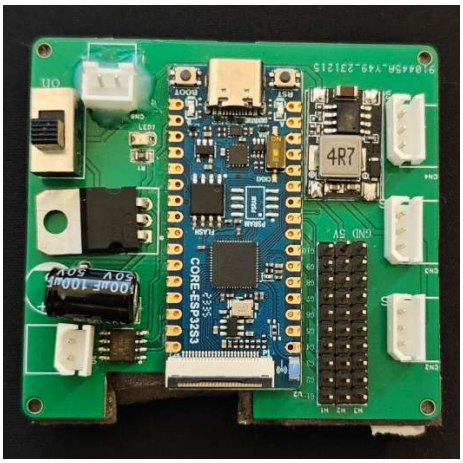

Figure 31 - PCB with every component

5 Software Design/ Programming

5.1 Overview - Arduino

In this project, I chose to use Arduino and its related libraries to control the robots. I used Arduino with specialized libraries and custom-defined header files and functions to achieve various functionalities, such as wireless communication between units, Bluetooth communication from devices to robotics, and accurate robot movements with PID control.

To write the program, compile, and upload the code, I chose Visual Studio Code and Platform IO, as they provide the best functionalities and versatility for this project.

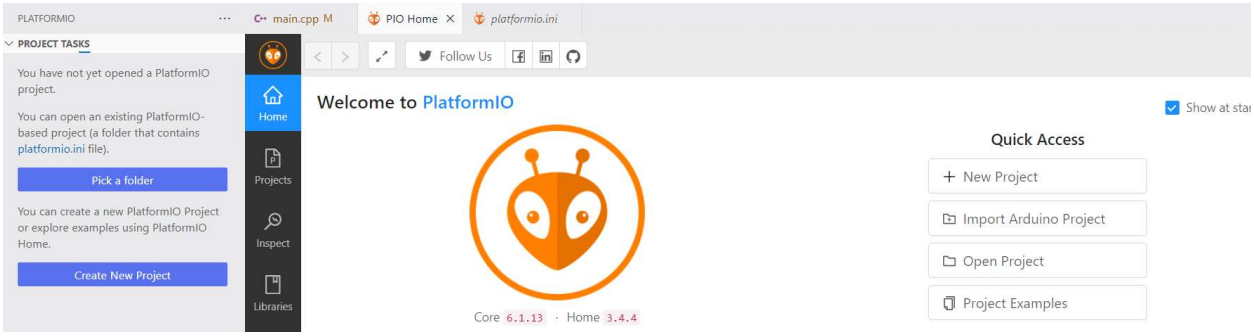

Figure 32 - Platform IO in Visual Studio Code

I also created two versions of the program and used version control. In the first version, my code is rather simple and aimed at achieving all the basic functionalities without having a too complicated program. In the second version, I revised most of the modules in the first version and added additional functionalities and features so that it would be more usable.

5.2 Wireless Communications

5.2.1 Unit-to-Unit Communications – V1

In a multi-unit robot, units must be able to communicate with each other to communicate movements and how each should react within a certain combined unit. With this in mind and given that this project uses ESP32S3 as the mainboard, I decided to use the “ESP Now” Library. This library allows unit-to-unit communication through WIFI and multiple channels with relatively low latency. In my program, I made one unit the center controller and then that unit will send data to every other unit, telling them how to move.

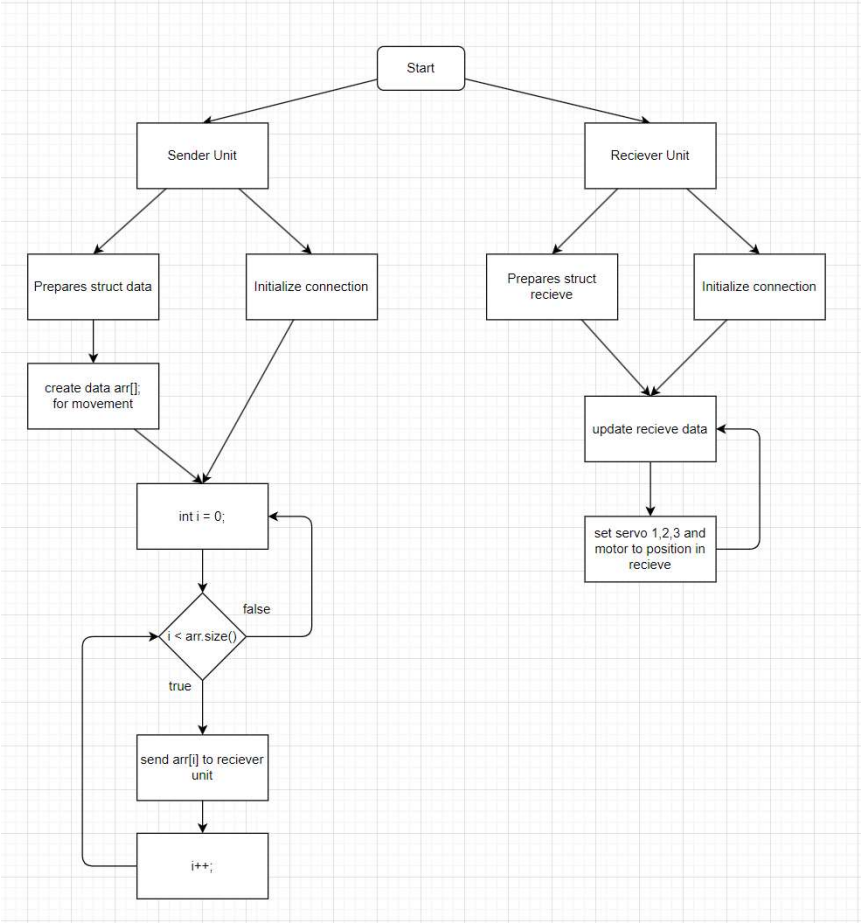

Figure 33 - Unit to Unit communication flowchart (single movement repetition)

5.2.2 Unit-to-Unit Communications – V2

In the second version of the program, I decided to continue using the "ESP Now" Library. However, I added a new class to package the send, receive, and initialization unit. It will automatically set itself to a specific channel upon initialization with a parameter. This dramatically improves the simplicity of the program as I only need to write this one class and then call it multiple times.

I also stored the mac address of peers within an array so that it can be called easier instead of writing specific mac addresses every time.

5.2.3 Device-to-Unit Communications – V1

In this project, I still want to be able to control the robot instead of letting it move forward by itself. Therefore, I need a way to transfer commands from my computer to the unit (so that it can move). In the first version of my program, I was only concerned about moving one unit at a time with my device. The way that I chose to do this is through a new library called "Blinker". This library allows even faster communication over Bluetooth. Although this approach still needs a mainboard as a relay, it is much more responsive than "ESP Now" and has a customizable built-in GUI. This approach further reduced the latency described in the previous approach and is also easy to use and implement.

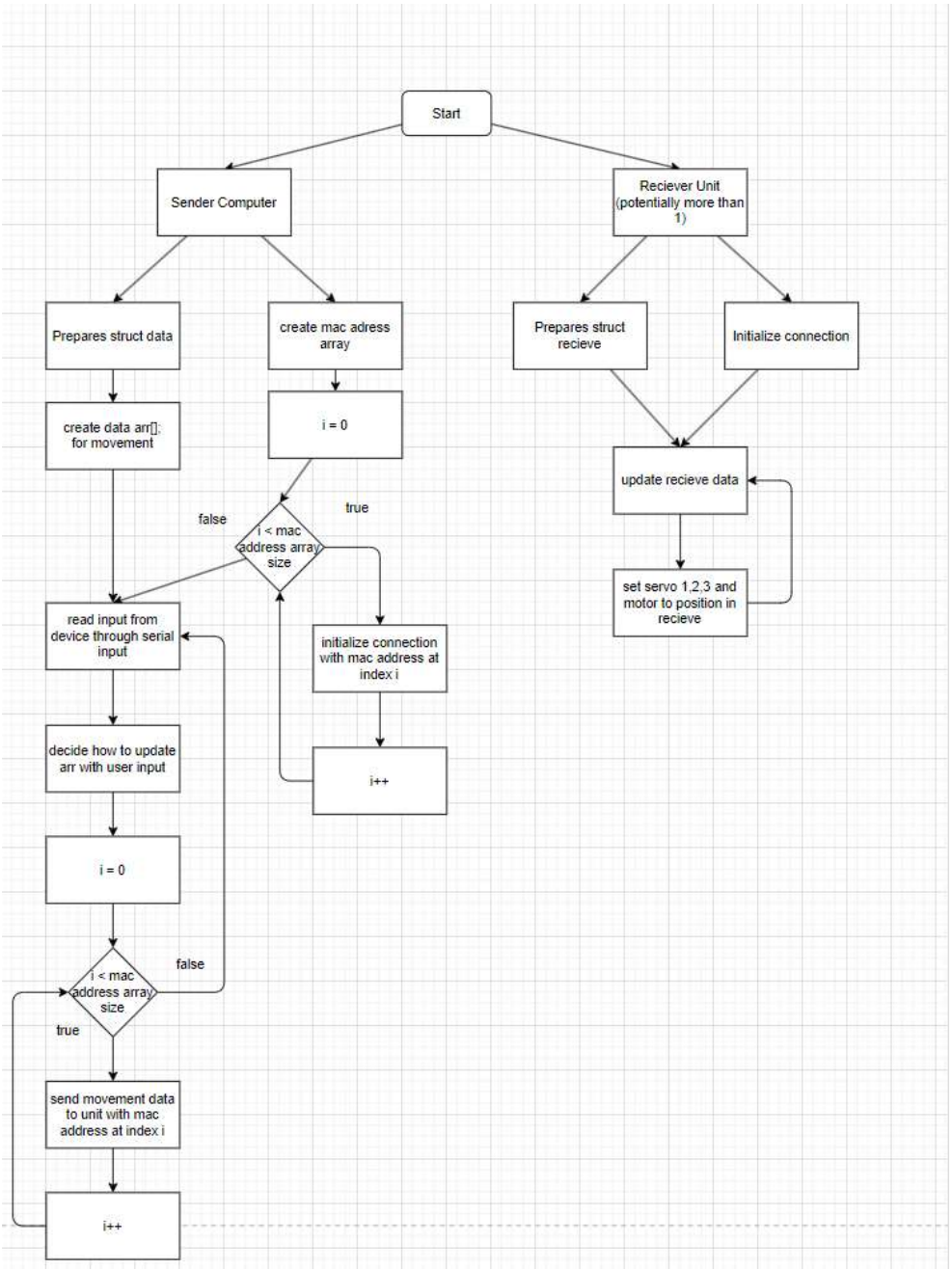

Figure 34 - Device to Unit communication V1 flow chart

In the flow chart, the sender establishes connections to multiple units. However, it only controls one, but for the convenience of switching which unit to control, all connections are established (or attempted because units might be powered off).

5.2.4 Device-to-Unit Communications – V2

However, I soon realized that there are actually no point in establishing connections to multiple units as it is not like I am switching between units for this purpose. Therefore, I decided to fall back to only controlling one unit at a time as well as changing the “ESP Now” library to Blinker (with Bluetooth). Theoretically, it can provide a lower response time and, thus, a snappier response.

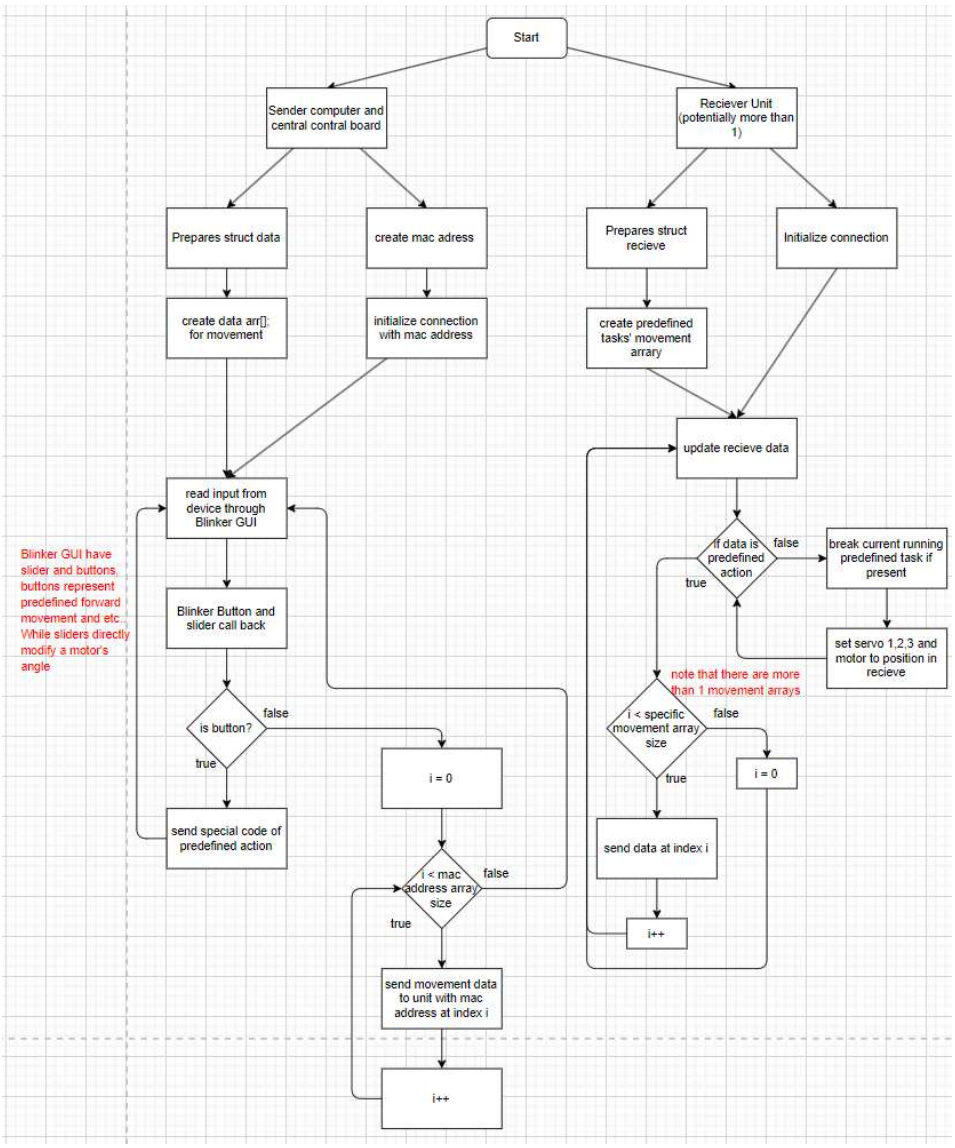

Figure 35 - Device to Unit communication V2 flow chart

5.2.5 Device-to-Multi-Units Communications

Then, I wanted to control more than one unit with my one device. One way to accomplish this is through the previously mentioned “ESP Now” Library. I was able to hook up an external mainboard and make it the center control unit. Then, this center control unit will be able to relay this information to every unit in the robot (like unit-to-unit communication in the previous section). Note that device-to-unit communication completely replaces the original unit-to-unit communications. The benefit of this method is that it is relatively simple to implement, as I already have a functioning “ESP Now” send and receive program on each mainboard. However, the downsides are that this introduced double the amount of delay, as the information is relayed twice throughout the entire system, and I still need to manually type each command into the Serial Input for the Arduino to understand (Which is potentially solvable with a GUI, but the method described in the next paragraph is much more efficient).

To reduce some of the latency and avoid writing an extra GUI, I decided to use “Blinker” to communicate between my device and the mainboard, similar to how the first version of the program worked.

However, the problem with this computer-to-mainboard-to-unit method has a drawback: the latency is high. Therefore, I was unable to have the unit break out of motion while doing it. Each motion simply does not last very long, and it would not have made a difference.

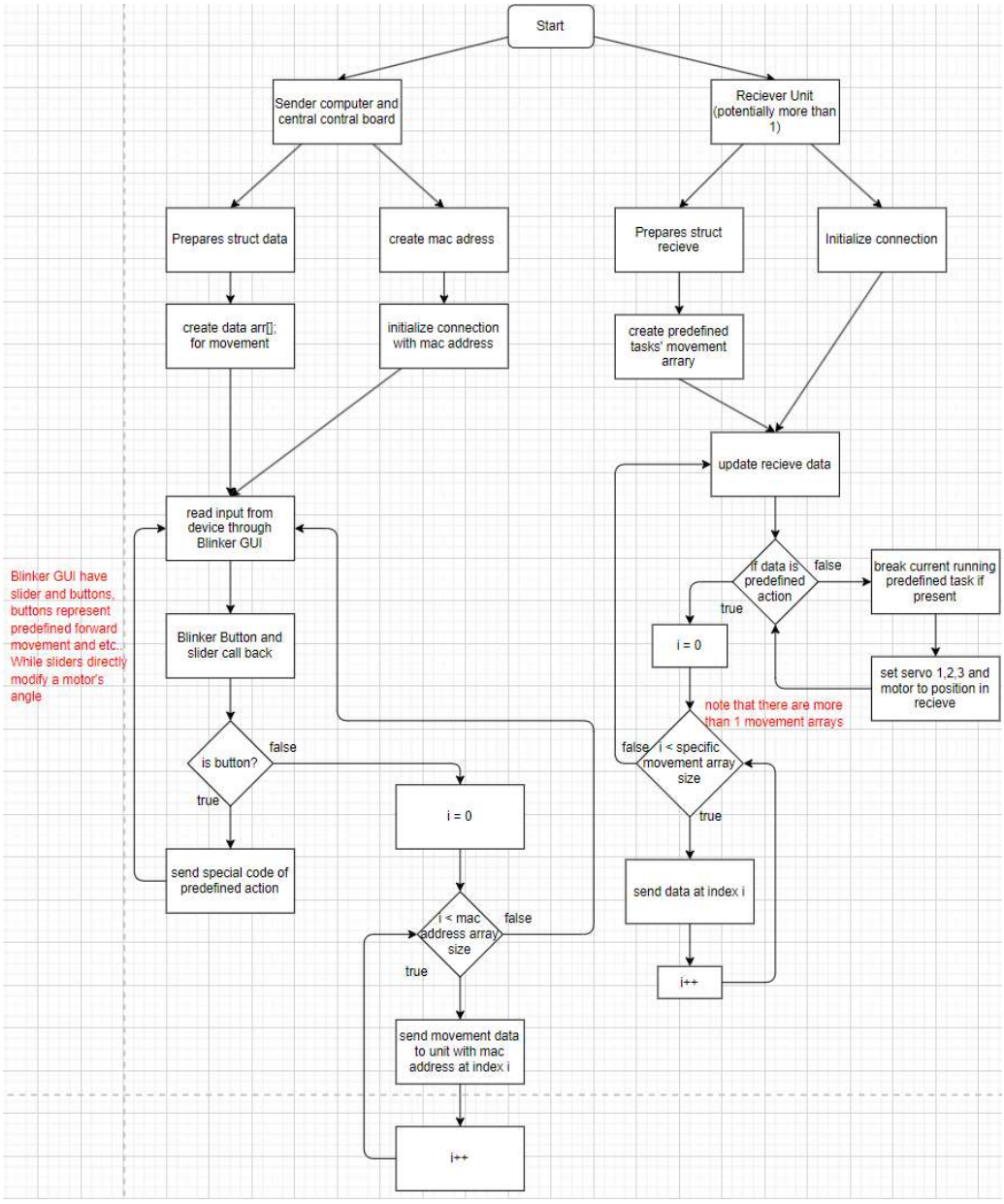

Figure 36 - Device to multiple units communication flow chart

5.3 Multitask Handling

5.3.1 Benefits of Multitask Handling

To properly illustrate the benefits of multitasking, we can imagine our normal computer. It is great how we can run multiple applications all at once. This is what is known as concurrency. Modern computers can achieve this by utilizing multiple cores and having multithreading turned on. However, in most Arduino mainboards, such as the ESP32S3, it only has 1 core and 1 available thread. This means that under normal circumstances, only one execution can be run at the same time. Therefore, we cannot run three commands that all take 1 second to complete at the same time. However, through something called FreeRTOS, we can achieve a similar functionality.

5.3.2 Achieving Simultaneous Execution

To handle multiple tasks simultaneously, I used a built-in functionality in the ESP32S3 mainboard: FreeRTOS. FreeRTOS allows custom tasks to be defined to run almost simultaneously, with certain limitations. In the program, I defined three custom tasks, each to control a servo based on remote data, alongside the data reception task running in the default “void loop()” task. This handling allows responsive actions from each unit, not including data transmission delays.

In order to program the ESP32-S3 with FreeRTOS, I first include the required libraries and then define the following function that will be run. After defining this function, I used the FreeRTOS command to start this function (also commonly referred to as a task) during “void setup” and set each to its respective priority and required memory. Detailed implementation will be discussed in Section 5.4.3.

5.4 Movement Control

5.4.1 From-Device Movement Control V1 & V2

When the controller wants the robot to perform certain tasks/movements, they can use the abovementioned Device-to-Unit communication method (Section 5.2.2). Operations through the Blinker GUI will be wirelessly communicated to the center control board. I used the following code to receive data from the Blinker GUI:

```
BlinkerSlider Slider1("SliderKey1");
BlinkerSlider Slider2("SliderKey2");
BlinkerSlider Slider3("SliderKey3");
BlinkerSlider Slider4("which");
BlinkerButton Button1("ForwardKey");
BlinkerButton Button2("BackwardKey");
BlinkerButton Button3("reset");
BlinkerButton Button4("lock");
BlinkerButton Button5("lock2");
BlinkerButton Button6("switch");
BlinkerButton ForwardBtn("forward");
BlinkerButton SpinBtn("spin");
BlinkerButton LFBtn("left forward");
BlinkerButton RFBtn("right forward");
BlinkerButton Move1Btn("move1");
```

Figure 37 - Blinker Component Creation

```
void slider1_callback(int32_t value)
{
    robot[num].a = value;
    esp_now_send(broadcastAddress[num], (uint8_t*)&robot[num], sizeof(robot[num]));
}
```

Figure 38 - Sample Blinker Component Handling Function Declaration

```
BLINKER_DEBUG.stream(Serial);
Blinker.begin();
Blinker.attachData(dataRead);

Slider1.attach(slider1_callback); Slider2.attach(slider2_callback); Slider3.attach(slider3_callback);
Slider4.attach(slider4_callback);

Button1.attach(button1_callback); Button2.attach(button2_callback); Button3.attach(button3_callback);
Button4.attach(button4_callback); Button5.attach(button5_callback);
Button6.attach(button6_callback);
ForwardBtn.attach(forward_callback); SpinBtn.attach(spin_callback); LFBtn.attach(fl_callback); RFBtn.attach
(fr_callback); Move1Btn.attach(move1_callback);
```

Figure 39 - Blinker Component Declaration

Then, the central control board will send data to the unit (or units), each containing the data for them to move. The ESP-Now program is directly called when pressing each button or slider, further decreasing the possible latency.

5.4.2 Data Packaging – V1

In the first version of the program, I was only concerned with driving one unit, so I simply transmitted one value at a time. This results in an extreme program. With the “Blinker” Library, I simply transmitted each slider value to the robot, which is enough to run it.

5.4.3 Data Packaging – V2

In the second version of the program, when sending remote data, all data are packaged through a custom "Struct" written similar to those in C++. This "Struct" contains a unique identifier of each action and, based on each unique identifier, data on the specific movement or to trigger certain pre-written actions in each unit. Here is the custom Struct containing the data:

```
struct data
{
    int a, b, c;
    int lock;
};
```

Figure 40 - Motor Data Structure

In this custom struct, integers a, b, and c each represent the position of the servos, and integer lock represents the state of the locking motor (for unit connections). Besides, to send data to all 4 units, I created an array of this custom structure. Therefore, I can directly change the structure within this array and send it to the respective unit. This implementation has a wider expandability than creating a single struct for each unit instead of creating an array. This approach allowed the use of a central controller with "ESP Now."

5.4.4 Motor Control – V1

In my program, motor control is achieved through 3 simultaneously running tasks (with FreeRTOS mentioned previously). Whenever new angles for these motors are received, the program sets it to three different variables. Then these 3 simultaneously running tasks will set the servo's degree to these updated values that's passed into the FreeRTOS program through their parameters.

5.4.5 Motor Control – V2

In the next version of the Motor Control program, I did not change too much of the core concept and how I achieved the motor control. The only change I made was that instead of passing the parameter into the FreeRTOS function every time, I initialized a global variable that contains the angle information of the servo. Within each FreeRTOS function, I simply set the motor degree to that global variable. This simplifies the program and allows other potential programs to change the servo's positions more easily, as they only need to access this variable.

Here is the function:

```
void task1(void* pvParameters)
{
    (void)pvParameters;
    while (true)
    {
        sA.write(posA);
        delay(3);
    }
}
```

Figure 41 - Motor Control Function Declaration

This is the FreeRTOS declaration:

```
xTaskCreate(task1, "some task 1", 2048, NULL, 2, NULL);
```

Figure 42 - FreeRTOS declaration

5.5 Unit Movements

5.5.1 Overview

Recall the design of each unit: two joints (one on each end of the unit) that can rotate in the same plane and one joint that can rotate in another plane.

Due to the significant number of limitations for a single unit, the only feasible option to move is by alternately rotating the two joints on each end. This creates variety in the friction on both ends, allowing the robot to shift slightly forward or backward.

5.5.2 Motion Control Program/ Process

In order to achieve an effective forward movement, many values, such as the degrees of rotations and their respective durations, have been tweaked to achieve optimal performance. Also, moving backward simply runs this sequence of movements in reverse order.

In order for the controller to execute this action, a button press on the Blinker GUI will trigger the function to send a signal to the central control. Then, the central control will be able to register this action and send an individual signal to the unit containing the degree to which each servo should be. These data are sent with the method described in motor control. The only difference is that the forward motion program has predefined values stored within an array. So, instead of the user putting values in one movement by one movement, the robot can move forward with one click.

Within each unit, the following function within the “void loop” receives the signal, and the FreeRTOS tasks will set the degree of the three servos to their respective value.

6 Experiments of Final Version

6.1 Single Unit Test

To test the ability of a single unit to move forward, we have to set the following criteria: it is moving on a smooth floor without any objects blocking its way. Besides, because of the limitations of a single unit and its movement style (by crawling forward), the floor will have a large impact on how much a single unit can move.

The single unit can move forward because of the variety of friction and the relatively smooth floor that I test on. In a single unit, I can change the friction at the front or back by rotating the two joints. On a rather smooth surface, when a joint is rotated 90 degrees such that the face is facing the floor, it has a small friction. However, if the joint is rotated 45 degrees so the face is slanted into the floor, it is nearly impossible to drag or push. Therefore, through changing between different states of the front and back joints, a single unit with only 2 joints that can rotate vertically to the surface can move forward and backward.

Figure 43 - Unit at Time 0.000s

This is the resting state of the robot. All joints are at 180 degrees and flat.

Figure 44 - Unit at Time 0.915s

At time 0.915 seconds, the bottom joint fully rotates to face the ground while the top joint had just started to rotate.

Around this stage the robot is being dragged forward again the friction. This is because of the state of the joints (and the faces touching the floor), the friction on the floor to be dragged forward is greater than the friction force.

Figure 45 - Unit at Time 1.149s

At 1.149 seconds, the bottom-most joint of the unit starts to reset to 180 degrees while the front continues to rotate to face the ground.

At this stage, the robot is being pushed forward against the friction.

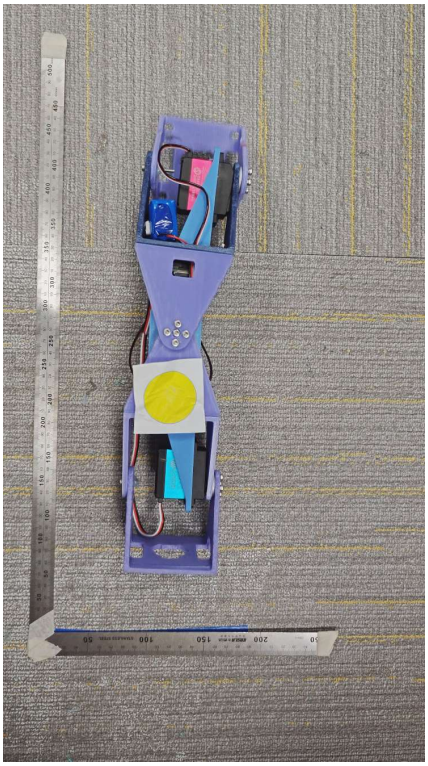

Figure 46 - Unit at Time 1.332

At time 1.332 seconds, the bottom joint completely resets to its beginning position while the top joint completely faces the ground at stays at its position.

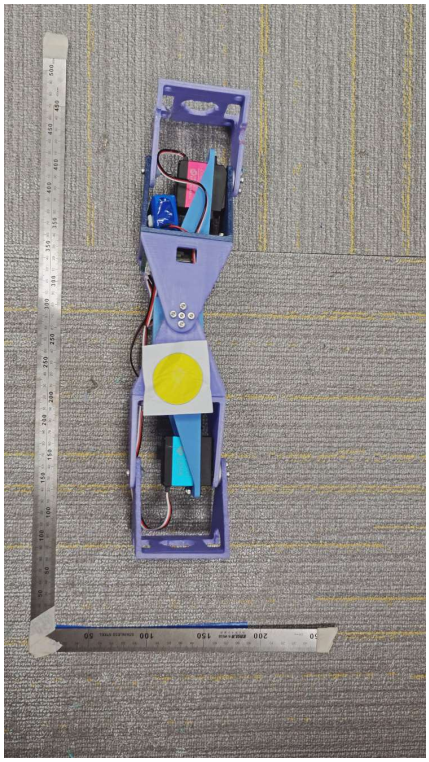

Figure 47 - Unit at Time 1.682s

At 1.682 seconds, the robot completely reset's its joints and the entire unit goes back to resting state while moved forward approximately 72 millimeters. (72 cm per cycle is an approximate average over all attempts, including the one shown above)

During the last stage and this stage, the unit is being slightly pushed backwards. However, due

Figure 48 - Single Unit Tracking

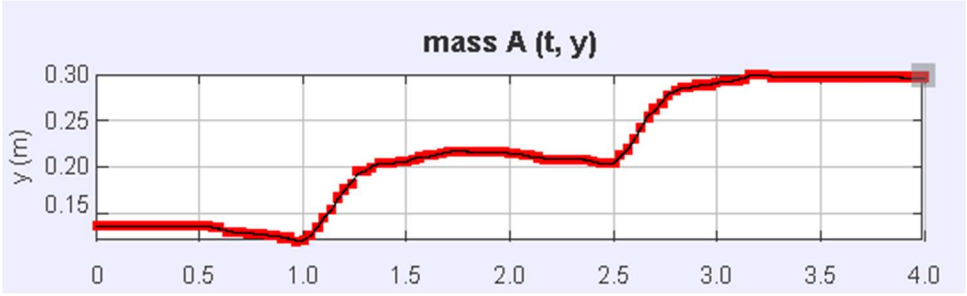

Figure 49 - Distance Tracking

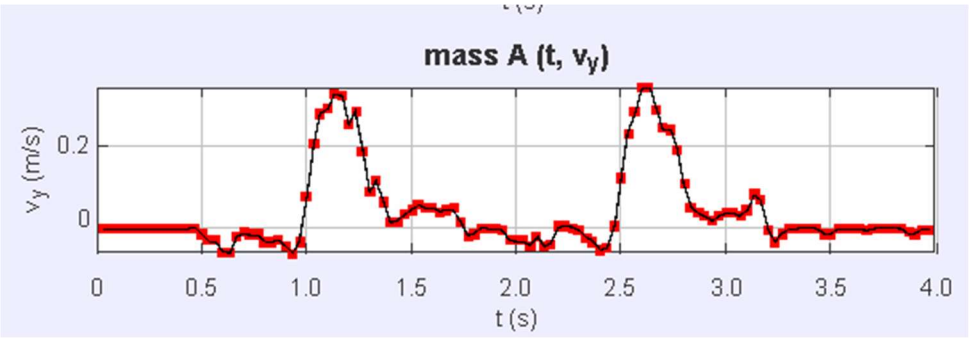

Figure 50 - Speed Tracking

After this sequence of movements, this unit moved forward around 72 millimeters over the span of 1.682 seconds. There is no need to perform a backward test as the unit is entirely symmetric over that axis, and the ability to move forward is the same as the ability to move backward.

6.2 Two Units Test

6.2.1 Movement Calculations

Overall, in two-unit movements, I will use a standing approach. Therefore, in order to maintain its balance, I need to calculate the locations and positions of both units. To do this, I can make use of trigonometry and coordinates to represent the entire robot in 3D space.

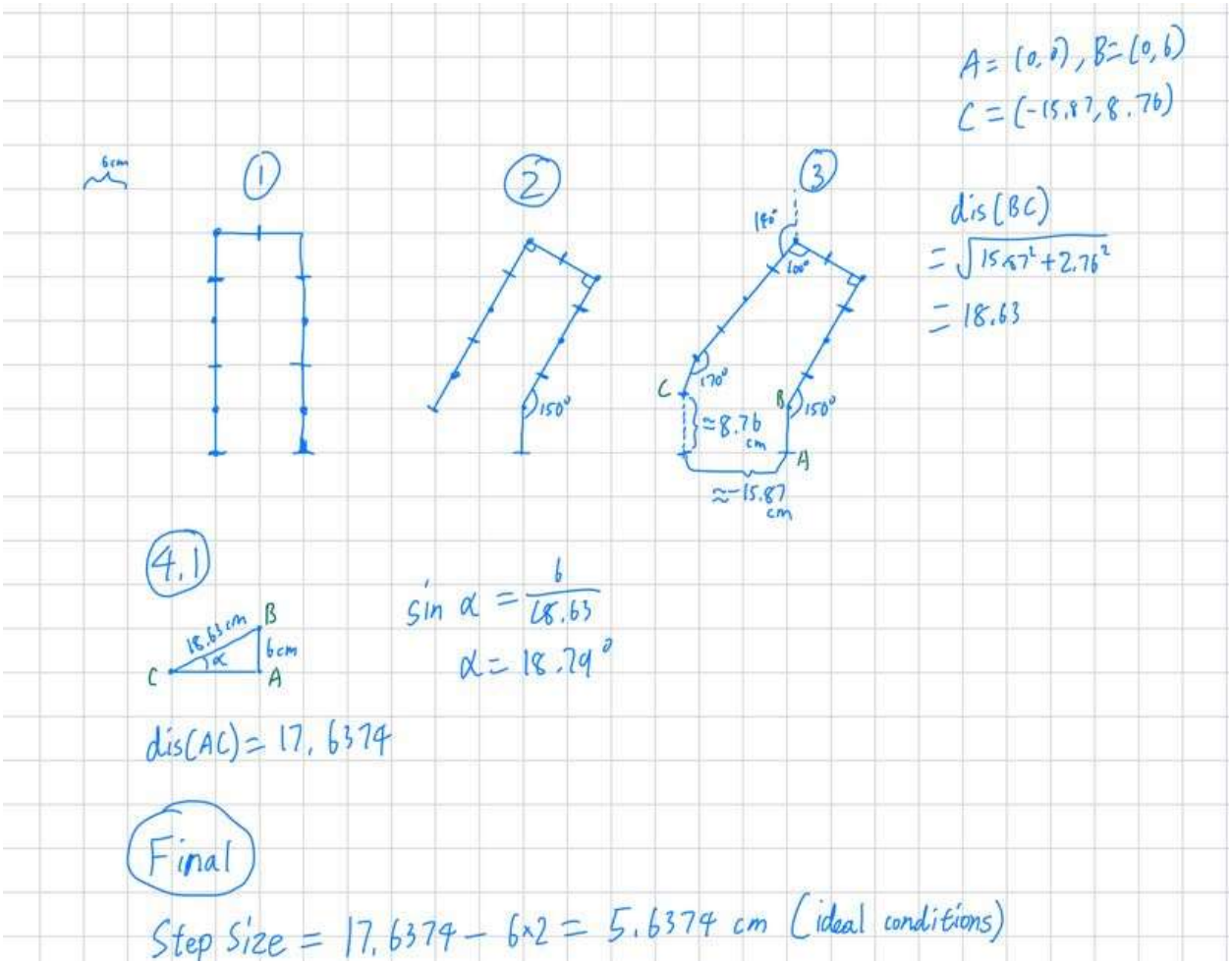

Figure 51 - Distance Calculations

In this calculation, each grid is 6 centimeters, a circle means a rotatable point, and a vertical line means connection between two distinct units. It uses trigonometry to calculate how far each step will make under ideal conditions with the current angles. Overall, each step can step forward about 5.6 centimeters every step.

6.2.2 Experimentation

After experimenting with a single unit's capabilities, we have to experiment with its capabilities when combined with other units. However, due to the special design of the joints, two units do not have as much better movement than a single unit.

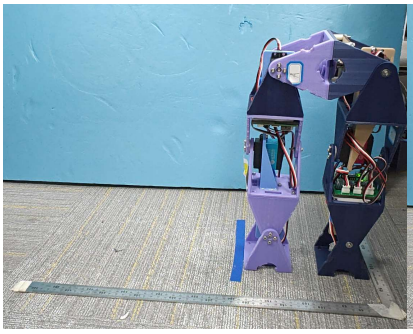

Figure 52 - units at Time 0s

For two units, I've decided to make it walk straight up. This position is attainable simply through rotating the top two joints from completely flat to these 90 degrees position. Therefore, this is its resting state at time 0.

Figure 53 - units at Time 1.116s

The first step of moving forward is to rotate the back joint such that the whole robot can shift its center of gravity towards the back. This all happens while the front joint starts to rotate.

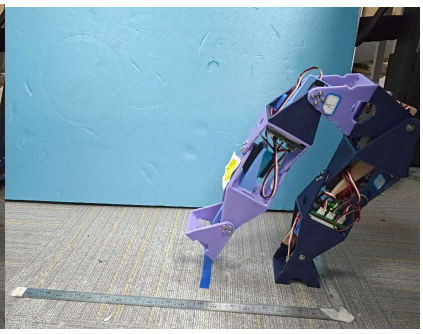

Figure 54 - Units at Time 2.398s

Now the whole robot have the entire center of mass at the back foot. Therefore, the front leg can start to extend forward to take a step.

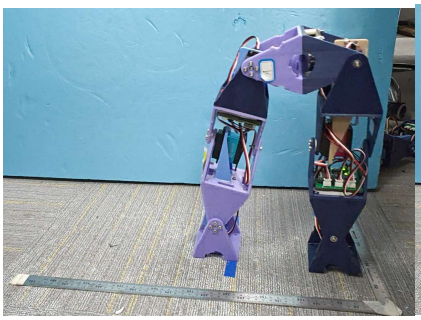

Figure 55 - Units at Time 3.863s

Afterwards the back leg pushes the entire robot forward to add to the distance that it travels in one step. Besides, this step is also necessary because we have to move the back leg forward (to an overall resting state).

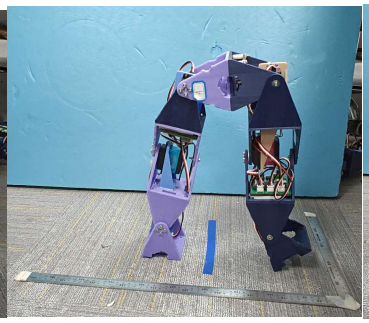

Figure 56 - Units at Time 4.513s

After about a second, the front leg will step down and completely touch the ground. Meanwhile, the back leg pushes the front leg forward so it can land properly.

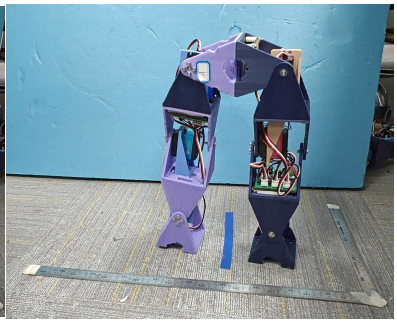

Figure 57 - Units at Time 7.828

Finally, the back leg gets dragged forward to back to a resting state. All of this process happens within 8 seconds and moves forward around 9 to 10 centimeters on average.

Overall, the steps are consistent at moving forward and are surprisingly stable, considering the fact that no extra balancing is done. However, one minor deficiency of this method of moving forward is that it potentially makes the robot end at an angle. Just in this example, while the robot is moving forward, it is also slightly rotating. Because of the lack of degrees of freedom and the need to keep the balance, it is difficult to adjust for the rotation.

6.3 Four Units Tests

As more units are attached together, more functionalities can be achieved. Now, it can swiftly rotate/turn on the spot, move forward in a straight line, and move forward on an angle.

6.3.1 Four Units Turn Tests

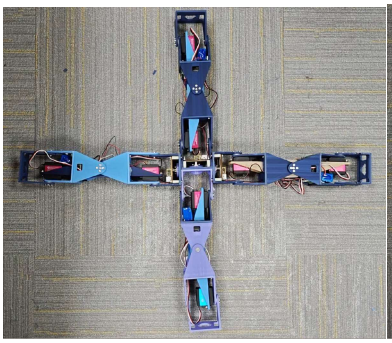

Figure 58 - Units at Time 0s

This is the initial resting state for the robot.

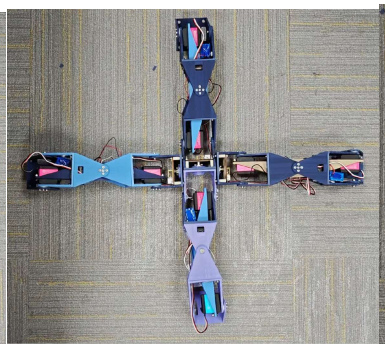

Figure 59 - Units at Time 0.677

The first step towards turning is to turn all of the outmost joints to facing the ground.

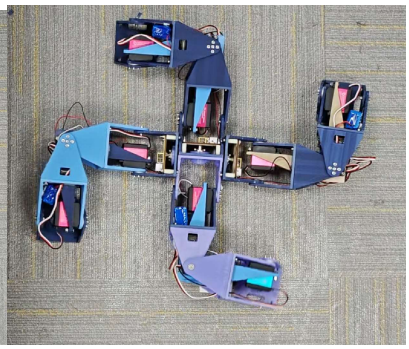

Figure 60 - Units at Time 1.428s

Then the robot bends all of the middle joints for the robot turn on the spot. It is worth noting that through changing the direction of these rotations and the degrees, we can control how much the robot the turns and to which direction.

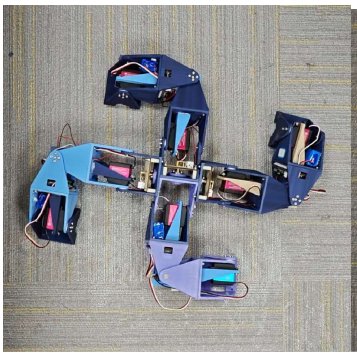

Figure 61 - Units at Time 1.845s

After, the robot will lift its center up so only the 4 outmost faces for on the ground.

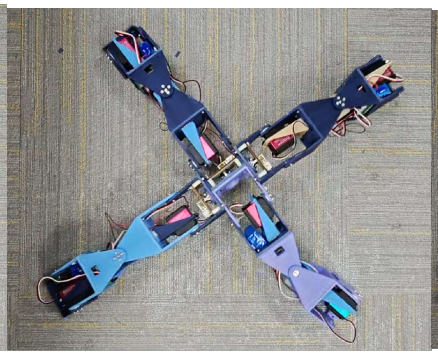

Figure 62 - Units at Time 2.827s

Then the robot flattens the center joints to rotate the robot.

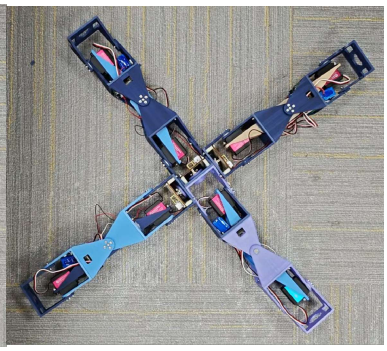

Figure 63 - Units at Time 3.877s

After moving the outmost joints back its initial positions, the robot is now again in resting position.

Overall, this on-the-spot turn is pretty good. It can turn about 45 degrees in around 4 seconds. This turning angle is also controllable by controlling how much the servo turns before lifting the center of the robot up. This approach is pretty effective for robots like this.

6.3.2 Four Units Forward Tests

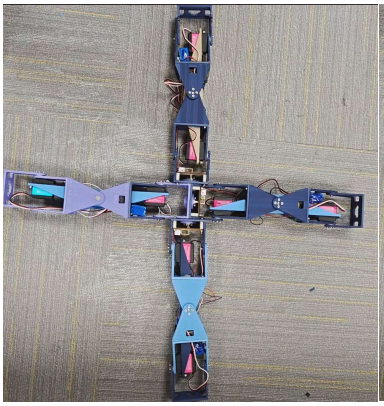

Figure 64 - Units at Time 0s

This is the initial resting position of the 4-unit robot.

Figure 65 - Units at Time 0.715s

Like rotating on the spot, the robot first turn all outmost joint to face the floor.

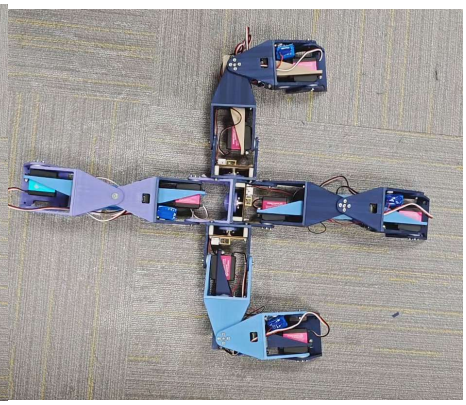

Figure 66 - Units at Time 1.267s

Unlike rotating on the spot, two arms on the side rotates 90 degrees to the direction that it is going to.

Figure 67 - Units at Time 2.168s

The robot than lift its center off of the floor and rotate both arms back straight, which moves the robot forward.

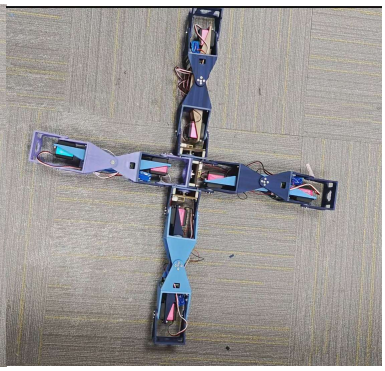

Figure 68 - Units at Time 2.749s

It then quickly rests all of the joints back to their resting positions.

Overall, the robot is capable of moving forward 14 centimeters within 3 seconds, which is quite effective. The amount of forward movement is also controllable by changing the degree to which the side arms are rotated forward.

6.4 Servo Strength Test

6.4.1 Overview

In this section, I measured the maximum weight that each servo within a single unit can lift (rotate). For convenience of the project. I will only be testing the capabilities of one joint, as mathematical equivalence could be drawn with a few formulas. Please do note that this testing data may vary based on batteries and how charged it is.

6.4.2 Servo Strength Results

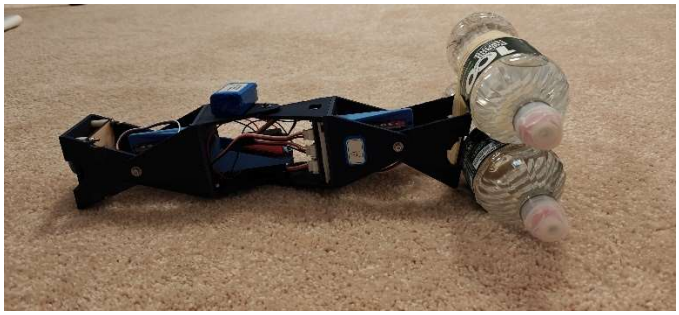

Figure 69 - Force Testing 1.4KG, Unraised

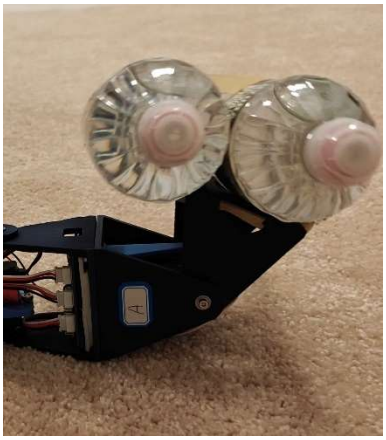

Figure 70 - Force Testing 1.4KG, Raised

0.7KG Weight	Passed
1.2KG Weight	Passed
1.4KG Weight	Passed
1.9KG Weight	Failed

Figure 71 - Force Testing Result Table

Finally, it's able to raise weight about 1.4 kilograms. From this result, we can tell that despite this joint doesn't have too much issue rotating with heavy items on its side. It's also not strong enough for this singular servo to rotate an entire 4-unit robot completely by itself, if given support.

6.4.3 Analysis

Overall, this joint can rotate despite weight and when multiple of them are working at once, can move robot pretty quickly. Besides, when the center mass of the rest of the component is further away from the servo, we know from physics that this servo will be able to rotate less weight (especially against gravity). However, because I'm still using a relatively weak servo, it cannot drive many units simply by itself. Therefore, this will limit some movements.

7 Conclusion

In conclusion, these sets of experiments demonstrated the abilities of this robot, whether it is usage in a single unit, two units, or four units. It is capable of moving in different directions and with different movement styles. Besides, as more time is invested into this project, more movement patterns and possibly more effective ones will be discovered, which fully demonstrates the potential of this project. Also, its expandability makes it perfect for dealing with unexpected events or conquering different terrains compared to those with wheels. In the future of this project, the controlling software between units can definitely be improved. Currently, as mentioned, they are only capable of moving in predefined groups. However, with improvements in software control, they are capable of much more, such as movement with different configurations and faster communication between units. Besides, the mechanical design can also be improved. It would most definitely benefit from properly machined components, as well as a stronger motor, a revamped joint design, and much more.

7.1 Future Outlook

In the future, this project can be expanded to include many aspects and be used in many different fields. For example, this type of robot can be made with smaller components to go through crevices in post-disaster scenarios. These robots can also solidify each joint and have stronger motors for it to be used as robotic arms in factories that require less precision. They can also be made with a variety of probes and sensors to map the world in the wildest of forests.

Besides modifications and add-ons to the physical structure, improvements can also be made to the software side. The current project utilizes a single center control board that, in fact, does not control a unit itself. If possible, we can make a mainboard on one of the units as the center control board, which mitigates the problem of needing an extra board. Besides, we can also allow units to make a “cluster,” and within each “cluster,” a board will be selected as the main control board. This approach alleviates the processing power required for the center control board but requires faster communication between these boards to sync their actions.

In the broader theme of application for this project, it’s clear that it has potential. For example, this system of modularity can be applied to robots that might require constant repairs from potential damage, as well as robots that requires wide adaptability from changing components. The base of modular robotics can also be joined with other innovations, such as soft robotics, to reach even wider possibilities.

8 References

[1] G. Song, Y. Zhou, Z. Wei and A. Song, A smart node architecture for adding mobility to wireless sensor networks, Sens Actuators A Phys, Vol. 147, No. 1, pp. 216-221, 2008.

[2] Xu, W., Han, L., Wang, X., Yuan, H., & Liang, B. (2020). Intelligent modularized reconfigurable mechanisms for robots: development and experiment. Research Square (Research Square). https://doi.org/10.21203/rs.3.rs-19729/v2

[3] Yang, C., Xu, B., Xia, J., Chang, H., Chen, X., & Ma, R. (2023). Mechanical behaviors of inter-module connections and assembled joints in modular steel buildings: A comprehensive review. Buildings, 13(7), 1727. https://doi.org/10.3390/buildings13071727

[4] Dorigo, M. (2005). SWARM-BOT: an experiment in swarm robotics. Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005. https://doi.org/10.1109/sis.2005.1501622

[5] Dorigo, M. (2014). The Swarm-bots and Swarmanoid experiments in swarm robotics. Dorigo - 2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC) - 2014. https://doi.org/10.1109/icarsc.2014.6849753

[6] Liu, C., Lin, Q., Kim, H., & Yim, M. (2022). SMORES-EP, a modular robot with parallel self-assembly. Autonomous Robots, 47(2), 211–228. https://doi.org/10.1007/s10514-022-10078-1

[7] Kuo, V., & Fitch, R. (2014). Scalable multi-radio communication in modular robots. Robotics and Autonomous Systems, 62(7), 1034–1046. https://doi.org/10.1016/j.robot.2013.08.007

[8] Luo, H., & Lam, T. L. (2023). Auto-Optimizing connection planning method for Chain-Type modular Self-Reconfiguration robots. IEEE Transactions on Robotics, 39(2), 1353–1372. https://doi.org/10.1109/tro.2022.3218992