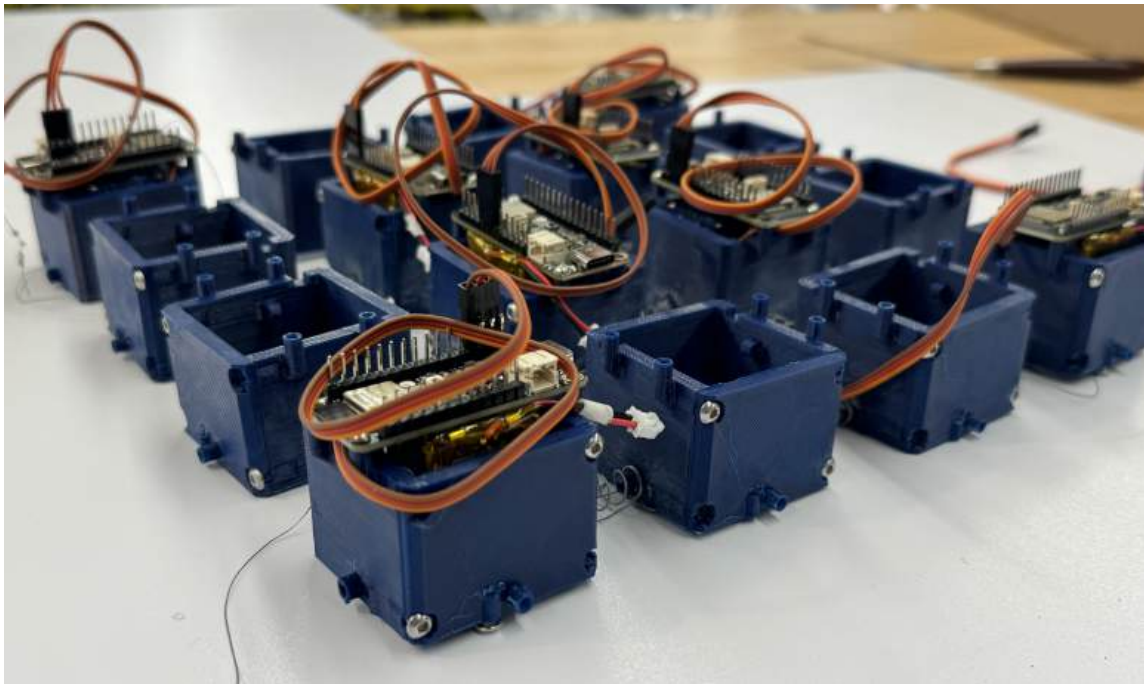


Robotic Metamaterial Documentation of Design

Maxwell Patwardhan

May, 2024



Contents

1	Introduction	7
2	Bill of Materials	8
3	Required Tools	9
4	System Assembly	10
5	Software & Networking	33
5.1	MQTT Broker	33
5.1.1	Installation & Setup	34
5.1.2	Configuration & Runtime	34
5.2	Embedded Software	38
5.2.1	Installation & Setup	38
5.2.2	Connection over Wifi	39
6	System Deployment:	40
7	Notes	41
7.1	Hardware	41
7.2	Bistable Mechanism	41

List of Tables

1	Bill of materials (BOM) robotic metamaterial	8
2	List of Necessary Tools and Electronics for Assembly and Use	9
3	Relevant PLA+ Print Settings	14
4	Relevant TPU Print settings. *Note: You may only use this setting with "Fast TPU" or Bambu Labs TPU. Other forms of TPU require a slower custom setting, which will be discussed below.	15
5	MCU-Servo Pin connections	29
6	'featherTest.ino' script command chart.	30
7	'mqttTest.py' script command chart.	32
8	Key Mosquitto Configuration Settings	35

List of Figures

1	Example layout for metamaterial. Blue nodes are active, and black nodes are passive. Subsequently, red lines represent driven connections, and black lines represent non-driven connections.	13
2	Example of one node sliced in Bambu Studio. This is the desired printing configuration for each part. Supports should only be generated for the inner overhanging portions of the node. Note that the pulley should be 3D printed in such a way that the part mating with the servo is facing upwards.	14
3	Example of Bistable Mechanism sliced in Bambu Studio. No supports are necessary when printing this component, but applying Elmer glue or nano-polymer glue is useful for bed adhesion. Notice that there is no infill for the arms, which limits how much they can be reduced further.	16

4	Bottom view of housing with highlighted edges that will be sharp when removed from the printer.	17
5	Housing and wall mated with two M3 screws.	18
6	Compression spring seated between nodes two nodes.	18
7	(a) Top of the pulley. The teeth interface with the gear of the servomotor. (b) Bottom of the pulley. This side of the pulley faces downwards when the robot is oriented normally.	19
8	Direction of threading through Pulley	20
9	1) Apply glue at the hole where the cable leaves the bottom of the pulley. 2) Apply glue to ‘tack down’ the loose end of the cable. . . .	21
10	An assembled active node with the Microcontroller and battery connected. Use this orientation for the electronics on each node.	22
11	Exploded view of an active node with all components. 1) Adafruit Feather microcontroller. 2) FeeTech Servo Motor. 3) The main body of the node housing serves as the mounting point for all components. 4) Cable pulley. 5) Compliant bistable mechanism. 6) Lithium polymer power supply battery. 7) Removable lid for assembly and repair. . . .	24
12	CAD Housing with labeled mounting points. 1) Mounts for Adafruit Feather microcontroller. 2) Mounting hold for the removable wall. 3) Mounting hole for compliant bistable mechanism. 4) Anchor point for incoming cables. 5) Mounting point for Servo. 6) Cable port for incoming and outgoing wire. The cable port also fixes the location of the compression spring.	25
13	Inter-node routing in metamaterial system.	26
14	Bottom-view of cable routing between nodes with termination at neighboring passive nodes.	27

15	(a) A profile section view of how the bistable mechanism is actuated. (b) An alternate view of how cables are routed through the bistable mechanism.	28
16	Heartbeat exchange between broker and 4 MCU's, addressed "esp32-client-#", on MQTT broker. The heartbeat frequency can be set in the embedded firmware. The heartbeat is a good visual confirmation of all clients on the network.	32
17	Client-Broker based structure of MQTT.	33
18	Terminal output of ' mosquitto -h '.	34
19	IPv4 address output when using the command ipconfig	36
20	Terminal output when using the command netstat -an which shows the state of the Port 1883 as "Listening".	37
21	Terminal output when using the command netstat -an which shows the state of the Port 1883 as "Listening".	37

Document Structure:

Section 1 gives a *creative* introduction. Sections 2 and 3 list the required materials and tools, respectively. Section 4 describes the overall process of assembling the system and then provides granular details for each step. Section 5 describes how to set up and run all software necessary for networking and robot control. Section 6 describes how to deploy the robot. Lastly, Section 7 contains any other important pieces of information regarding the mechanical, software, and electrical subsystems.

1 Introduction

Suppose you wash up on a desert island surrounded by microcontrollers, PLA, TPU, actuators, batteries, fishing line, and mounting hardware. You look around and discover a fully equipped engineering makerspace; strange, but in this day and age, is anything truly surprising? As you come to, you feel an *overwhelming* urge to design a robotic metamaterial, even if it is the last thing you do! After all, spending your final days stranded on an island fruitlessly searching for food is rather pedestrian; you yearn to meet absolution head-on! As this desert island's foremost robotics researcher, welcoming deliverance with open arms will not happen without having made an edificial contribution to your vocation. Luckily for you, this document is the key that will turn your diaspora of naturally-occurring-desert-island-materials into a fully-fledged robotic metamaterial.

Dumb hypothetical scenario aside, I have spent over five months working to design and control this robotic metamaterial system. It has gone through countless iterations and small tweaks, many of which are not visible when reading my thesis or working directly with the system itself. During this process, I have learned quite a few techniques and tricks to assemble and operate this system efficiently. As the requirements for my degree are now met, I shall soon be shuffling off this academic coil¹, which brings about a conundrum for future users of the robot. To prevent many of the small tokens of knowledge from being lost in the sands of time, I will try my best to leave them in this addendum to my thesis. This document exists to let any person manufacture, assemble, and control their very own metamaterial system. I will assume you have a rough knowledge of this system's function. If you don't, go read my thesis to get an understanding - I guarantee it will provide context that is missing in this document. Anyways, it is likely I will forget some details in the process of documenting this system, so if you are the fortunate undergraduate student tasked with using this system, be sure to use the GitHub² repository for all design documents, and feel free to reach out to my email³ with further questions.

¹Analogous to Shakespeare's intended meaning: I will be joining the workforce.

²github.com/MaxPatwardhan/multistableRoboticMetamaterial

³maxwellpatwardhan@gmail.com

2 Bill of Materials

Part:	Vendor:	Part No.:	Unit Cost (\$):	Quantity:	Total Cost (\$):
Feather ESP32 V2	Adafruit	5400	19.95	1	19.95
350mAh 3.7V LiPo Battery	Adafruit	4237	5.95	1	5.95
FS90R Servo Motor	Adafruit	154	11.95	1	11.95
M3 × 6.00mm Screw	McMaster Carr	92095A179	0.0583	8	0.47
M2 × 6.00mm Screw	McMaster Carr	92095A454	0.2732	10	2.73
Compression Spring, 1" × 0.3 OD	McMaster Carr	9657K301	0.965	4	3.86
10 Lb. Fishing Line	Amazon	Find a cheap one.	0.02 (\$/ft)	4 ft	0.08
Matte PLA+	Overture	Matte Black	0.019 (\$/g)	16.83 g	0.32
Fast TPU	Overture	Clear	0.032 (\$/g)	1.75 g	0.06
Total:					45.37

Table 1: Bill of materials (BOM) robotic metamaterial

3 Required Tools

Required Tools & Electronics:	Purpose:
2 mm Hex Key	Driving M3 Screws
1.3 mm Hex Key	Driving M2 Screws
Phillips Screwdriver	Driving M2.5 Screw
Super Glue	Compression spring mounting
CA Accelerator	Allowing the glue to fast-dry
Double Sided Velcro	Battery mounting
Pliers/Cutters	Detailing & finishing housing
File/Deburring Tool	Deburring housing
Tweezers	Cable routing
USB-C Cable	Programming MCU & charging battery
WiFi Router	Wireless communication

Table 2: List of Necessary Tools and Electronics for Assembly and Use

4 System Assembly

This section will step through how to design your own metamaterial system. You will first be given an overview of what is required to build your own system, after which each step will be discussed in further detail. All mechanical components were designed in the student edition of Solidworks. Assembly and part files are provided in the CAD fold of the GitHub⁴ repository, though it is important to note that the assembly exists primarily for visualization purposes. Within said CAD folder, ‘.step’ files are also included, so you can quickly print any of the desired parts.

⁴github.com/MaxPatwardhan/multistableRoboticMetamaterial/blob/main/CAD.zip

System Assembly Process Overview

- Step 1: **Gather Materials:** Collect all necessary materials as specified in Sections 2 and 3. Determine desired node and cable tensioning configuration.
- Step 2: **Print Housings and Pulleys:** Print the desired quantity of housings and pulleys out of Polylactic Acid (PLA) or a material with equivalent properties.
- Step 3: **Print Compliant Mechanism:** Print the desired quantity of bistable mechanisms out of Thermal Polyurethane (TPU) or Polypropylene (PP).
- Step 4: **Finish Prints:** Remove supports and file away sharp angles to prevent cables from fraying. Mount the removable walls to the node body.
- Step 5: **Layout & Glue Nodes:** Layout nodes in desired configuration. Use superglue to place compression springs between nodes.
- Step 6: **Assemble Pulleys:** Thread cables through pulley holes. Glue the thread to the pulleys. Consider using CA Accelerator to drastically improve drying time.
- Step 7: **Wait for glue to dry:** A favorite pastime among America's elderly.
- Step 8: **Mount Components:** Mount the servo, the compliant bistable mechanism, and the battery in that respective order.
- Step 9: **Mount Pulleys & Route Cables:** Mount the pulleys onto the servo. Use tweezers and a respectable amount of patience to route cables through the compression springs and the bistable mechanism. Anchor the cables at opposing nodes with an M3 screw.
- Step 10: **Upload Embedded Test Firmware & Charge Electronics** Connect the battery to the microcontroller. Then, plug the microcontroller into your computer to allow the battery to charge. This should take roughly thirty to forty minutes. Using the Arduino IDE, upload the test script 'featherTest.ino' provided on GitHub.
- Step 11: **Mount & Wire MCU:** Mount the microcontroller to the node body and wire it to the servo.

- Step 12: **Test & Adjust Tensioning:** Drive each active node and ensure that it is tensioning its neighboring nodes properly. Adjust cable tensioning iteratively as needed.
- Step 13: **Upload MQTT Control Test Code:** Using the Arduino IDE, upload the provided MQTT test code. Be sure to match the mirror to the MCU's designated address in the code. Set the network and broker parameters accordingly.
- Step 14: **Run the MQTT Broker:** Run the Mosquitto MQTT broker and ensure that clients can connect to the network.
- Step 15: **System Test:** Run the 'mqttTest.ino' and 'mqttTest.py' scripts on the respective platforms and make sure nodes are receiving and performing commands as desired.

Step 1: Gather Materials

Use the bill of materials 2 and the list of required tools 3 to gather all necessary components for building the robotic system. Presumably, you are a member of the CRB and have access to the makerspace. I would recommend searching for many of these tools and materials there before going and buying them on your own. At this point, you should also have a clear idea of how you want to configure the nodes. Are you making a 3×3 grid? A 4×4 grid? Are you making a trapezoid? Whatever you decide to build will determine how many active nodes (nodes with microcontrollers, servos, etc.) you will need. Once you have determined the quantity of active nodes, count how many mentioned connections you would like - this will come in handy later. In my time experimenting with this system, I have often found that placing active nodes on the perimeter of the robot, as shown in Figure 1, results in interesting and somewhat predictable behavior, making it an approachable way to begin controlling the system.

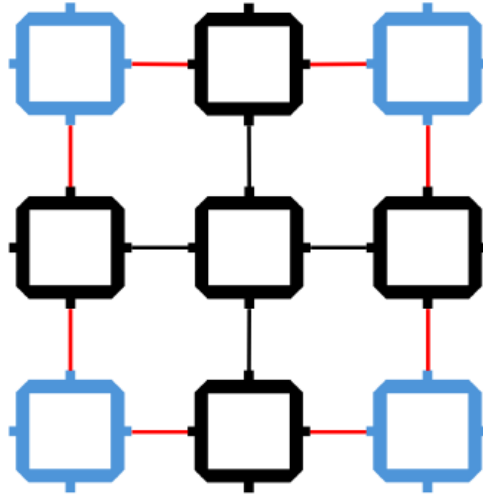


Figure 1: Example layout for metamaterial. Blue nodes are active, and black nodes are passive. Subsequently, red lines represent driven connections, and black lines represent non-driven connections.

Step 2: Print Housings and Pulleys

Before printing, it's a good idea to do a test print with the filament you plan to use. Manufacturing tolerances can vary depending on factors like filament color or the specific printer being used. This will impact how (and if) mounting hardware taps and threads into the printed components. Print a single housing, pulley, and removable wall to ensure all holes align correctly. Make sure you can drive the M2 and M3 screws into the housing, and ensure that the pulley sufficiently mates with the servo. Performing this test will save you hours of time and will prevent you from potentially wasting a significant amount of filament.

The housings and pulleys are printed from PLA on a Bambu X1 Carbon or a Creality K1 Max, and the Bambu or Creality slicers are used, respectively. Table 3 gives the relevant print settings for printing with PLA:

Print Settings: PLA	
Filament Setting:	eSun PLA+
Extruder Temp. (C):	235
Bed Temp. (C):	35
Support Style:	Tree (Auto)
Layer Height (mm):	0.08
Initial Layer Height (mm):	0.2

Table 3: Relevant PLA+ Print Settings

The values above are default in both the Bambu and Creality slicers. Print these components at the maximum Note that printing at printing an entire system will take a considerable amount of time at this resolution. I find that printing 16 nodes takes roughly 36 hours. Automatically generated tree supports are necessary for accurately printing the main housing; none of the other parts require supports. Figure 2 displays what each of the sliced parts will look like in the Bambu Studio.

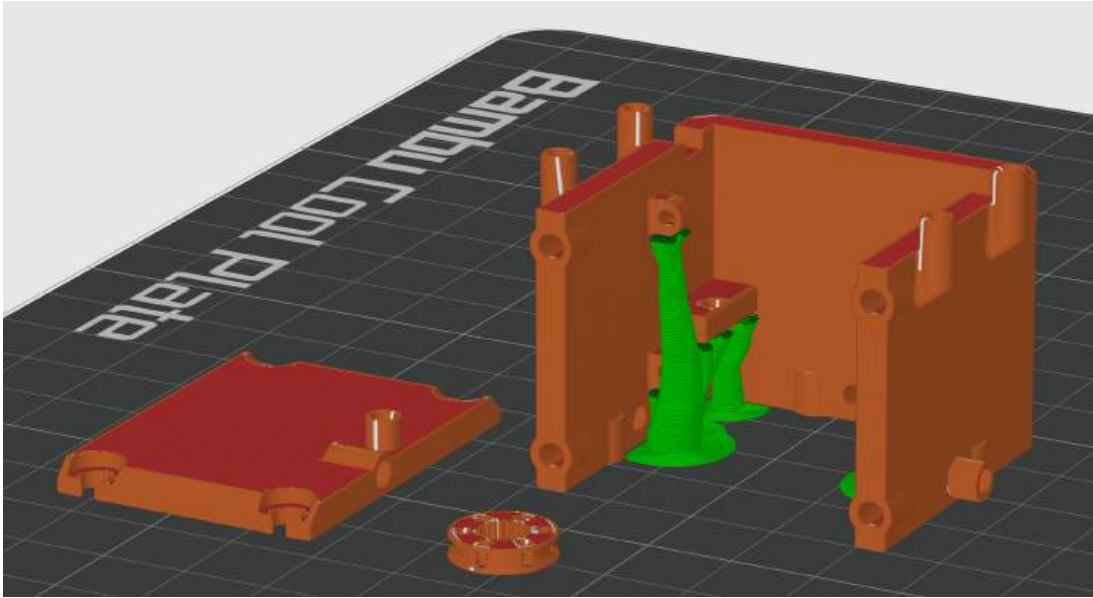


Figure 2: Example of one node sliced in Bambu Studio. This is the desired printing configuration for each part. Supports should only be generated for the inner over-hanging portions of the node. Note that the pulley should be 3D printed in such a way that the part mating with the servo is facing upwards.

Step 3: Print Bistable Compliant Mechanism

The bistable compliant mechanism is printed using TPU 95A. Here are the relevant print settings:

Print Settings: TPU 95A	
Filament Setting:	Bambu TPU 95A *
Extruder Temp. (C):	230
Bed Temp. (C):	30
Support Style:	None
Layer Height (mm):	0.08
Initial Layer Height (mm):	0.2

Table 4: Relevant TPU Print settings. ***Note:** You may only use this setting with "Fast TPU" or Bambu Labs TPU. Other forms of TPU require a slower custom setting, which will be discussed below.

To print the bistable compliant mechanism, "Fast TPU 95A" is used. The Bambu X1 Carbon is the only printer capable of doing this within the CRB makerspace. It is important to note that you can not use the AMS when printing with TPU and that you must load the filament from the back of the printer. Here⁵ is a good guide in case you are unfamiliar with how to set up this particular type of print. Note that with this specific type of TPU, you can use Bambu's default TPU settings. You must use a TPU rated at 95A stiffness or higher, and if you decide to use a TPU that isn't labeled as 'Fast', you will need to input custom filament settings in the Bambu Slicer; a good guide for this can be found here⁶. It is important to keep the door open when printing with TPU, otherwise the printer can not clear heat away fast enough with it closed. I find that one of these mechanisms takes roughly 15 minutes to print. Figure 3 shows a sliced version of the bistable compliant mechanism. The stiffness, k_B , of the spring mechanism is dependent on the thickness of its arms. There is little tolerance to decrease the thickness of the arms of the bistable spring, so it is important to keep this in mind when altering the properties of the bistable compliant mechanism in CAD. Tuning k_B will be discussed in greater depth in Section 7.

⁵<https://www.youtube.com/watch?v=fhu5C-3AJvo>

⁶<https://wiki.bambulab.com/en/knowledge-sharing/tpu-printing-guide>

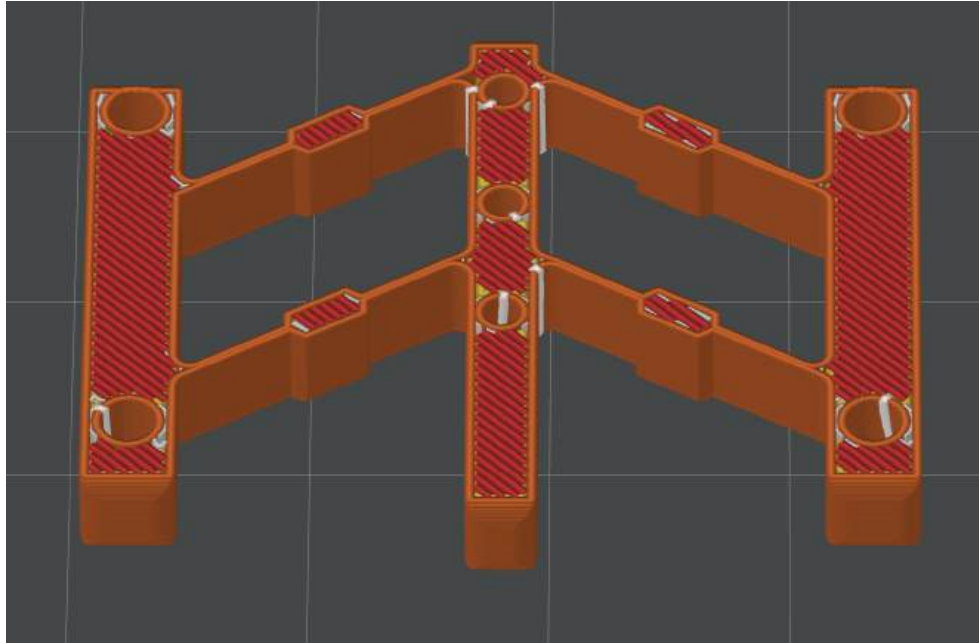


Figure 3: Example of Bistable Mechanism sliced in Bambu Studio. No supports are necessary when printing this component, but applying Elmer glue or nano-polymer glue is useful for bed adhesion. Notice that there is no infill for the arms, which limits how much they can be reduced further.

Step 4: Finish Prints

When the housing, removable wall, and pulleys have finished printing, remove them from the build plate. Now is a good time to ensure that everything fits together. Namely, ensure that the mounting holes of the removable wall are concentric with the mounting holes on the housing and check that the metric machine screws fit into respective holes if you have not already.

The initial layer of the PLA prints is 0.2 mm thick for adhesion while printing. This creates sharp edges at the bottom of the housing and removable wall, causing the tensioning cables to fray over time. Use a deburring tool, exacto knife, or file to smooth the bottom edges of the housing, which will preserve the tensioning cables. Figure 4 highlights which edges specifically need to be deburred.

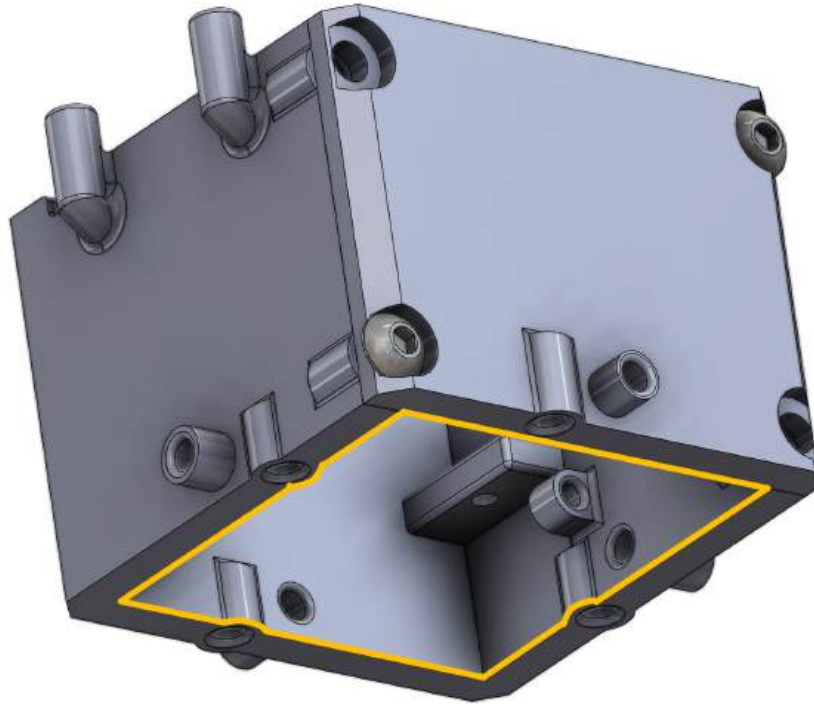


Figure 4: Bottom view of housing with highlighted edges that will be sharp when removed from the printer.

Step 5: Layout & Glue Nodes

Now that the housings have all been finished, you can attach the removable walls to the housings using two of the M3 screws, as shown in Figure 5

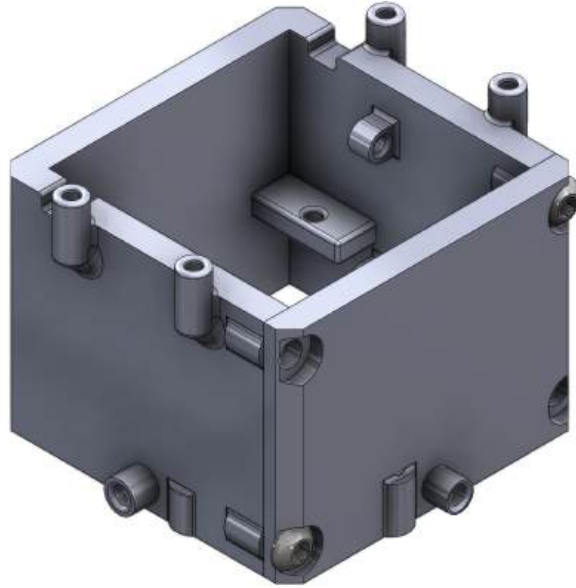


Figure 5: Housing and wall mated with two M3 screws.

Once all of the complete housings have been assembled, they should be arranged in the predetermined layout. Be sure to orient the removable wall towards the perimeter of the system, as that will make it easier to access and assemble the active nodes.

Once the nodes are arranged, seat a compression spring around the protruded routing holes for the cable between nodes, as depicted in Figure 6.

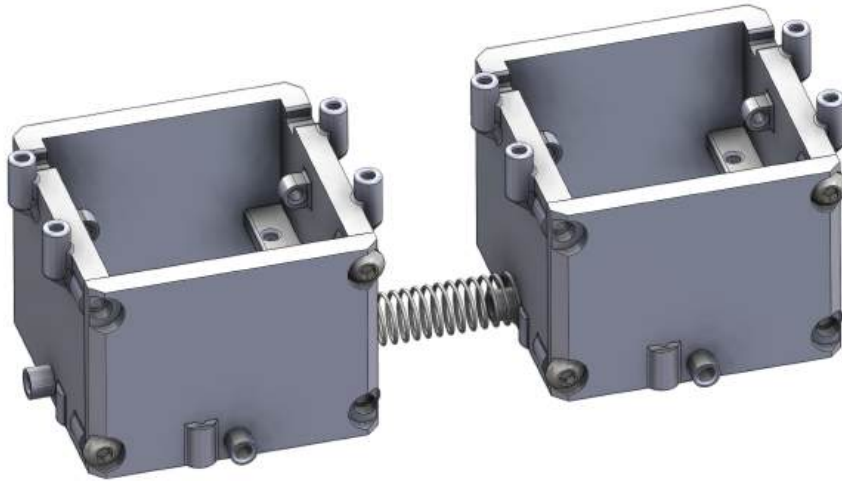


Figure 6: Compression spring seated between nodes two nodes.

Seat a compression spring for every connection in your layout. You are essentially assembling the structure of your system without any of it fastened yet. This is a delicate and time-consuming process, but it is important that you get the nodes and springs arranged evenly as you are about to glue everything in place. Apply a healthy dab of superglue to the point where the spring meets the housing, and ensure the alignment of both the node and springs. At this point, you can apply CA Accelerator to each joint, which will cause the glue to dry in a matter of seconds. If you do not use CA Accelerator, you should wait for roughly an hour. Once the glue has dried enough to handle the system, flip the grid upside-down and add another drop of superglue to the joints, followed with CA Accelerator if possible.

Step 6: Assemble Pulleys

Now, it is time to assemble the pulleys. You should have one pulley for every active node in your system. Cut one eight-inch segment of fishing line for each of the driven bistable compliant mechanisms, as well as one segment for each driven connection between nodes in your system. For example, if an active node is tensioning two neighboring passive nodes, as well as the internal bistable mechanism, you will need to cut three pieces of cable for this node. Figure 7 establishes orientation for the pulleys.

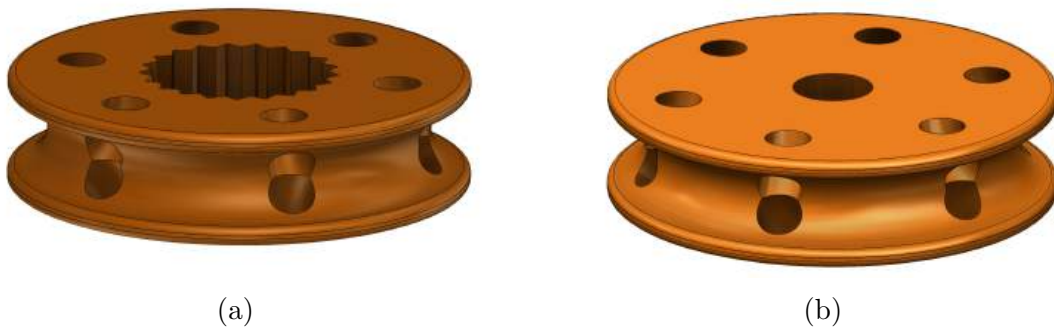


Figure 7: (a) **Top** of the pulley. The teeth interface with the gear of the servomotor. (b) **Bottom** of the pulley. This side of the pulley faces downwards when the robot is oriented normally.

Once you have cut the requisite amount of fishing line, thread it through each

respective pulley with equal spacing between holes. An example of how to orient one wire is shown in Figure 8.

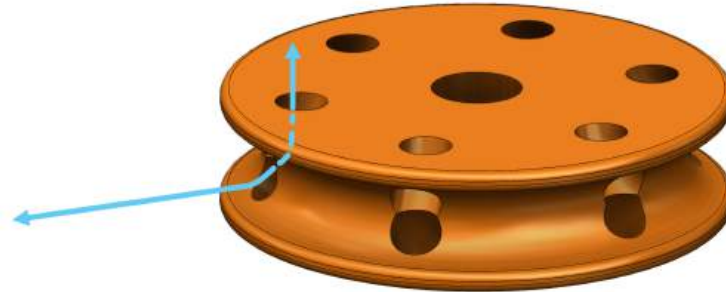


Figure 8: Direction of threading through Pulley

Tweezers are quite helpful for routing cables through the pulley holes. Once you have each cable arranged as described, apply a small amount of superglue to where the cable emerges from the bottom face of the pulley. I tend to add an additional bit to ‘tack down’ the end of the cable. Apply CA Accelerator if possible. Figure 9 shows the underside of an assembled active node with labeled areas where you should apply superglue.

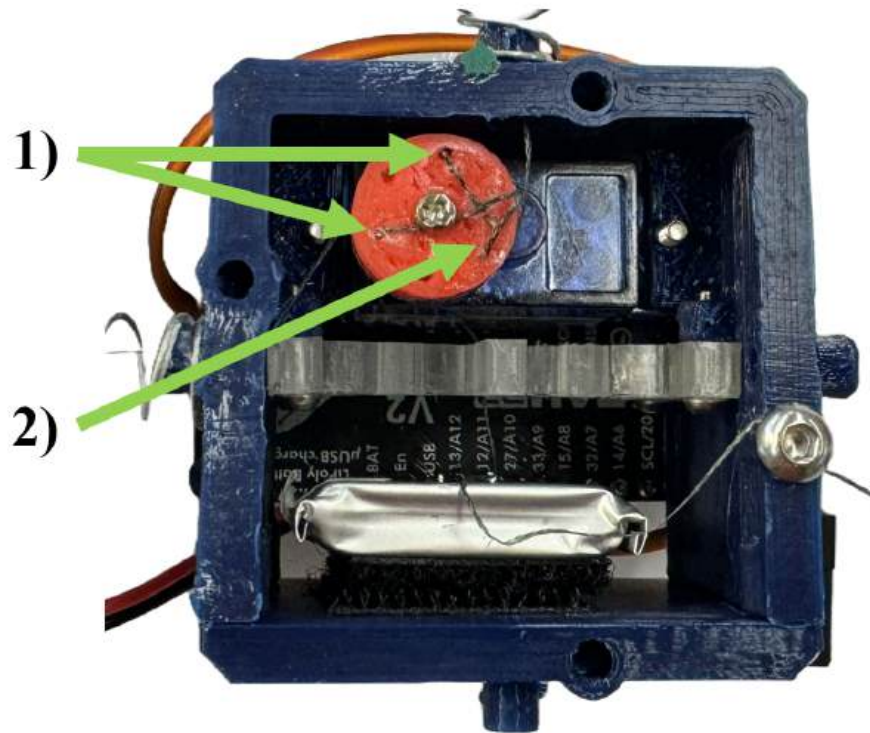


Figure 9: 1) Apply glue at the hole where the cable leaves the bottom of the pulley.
2) Apply glue to 'tack down' the loose end of the cable.

Step 7: Wait for Glue to Dry

Ponder your existence. What brought you here? How do they even make superglue? What sorcery do they employ to keep the process equipment from sticking to itself? How can a loving god cause such agony?

Step 8: Mount components

At this point, you can mount the servo, bistable compliant mechanism, and battery in that order, but do not mount the pulleys or microcontroller yet! You will need to turn the system upside down in order to mount and route the pulleys, and you don't want to put any unnecessary load on the microcontrollers while doing so. To mount the

rest of the components, you will need to remove the walls of each active node. This will be more challenging for active nodes that are surrounded on all sides, but simply bend the removable wall out of the way when installing components. All mounting points on the housing are self-tapping. The battery is attached to the removable lid of the node using adhesive-backed Velcro. This provides secure yet gentle mounting for the Lithium Polymer cells, which are typically quite sensitive to being mounted against rigid objects.

Important: The battery connector is quite short, so it is necessary to mount the microcontroller and battery in the orientation that allows their connectors to reach each other. Figure 10 shows an assembled active node. Notice the orientation of the Feather microcontroller and the battery connector.

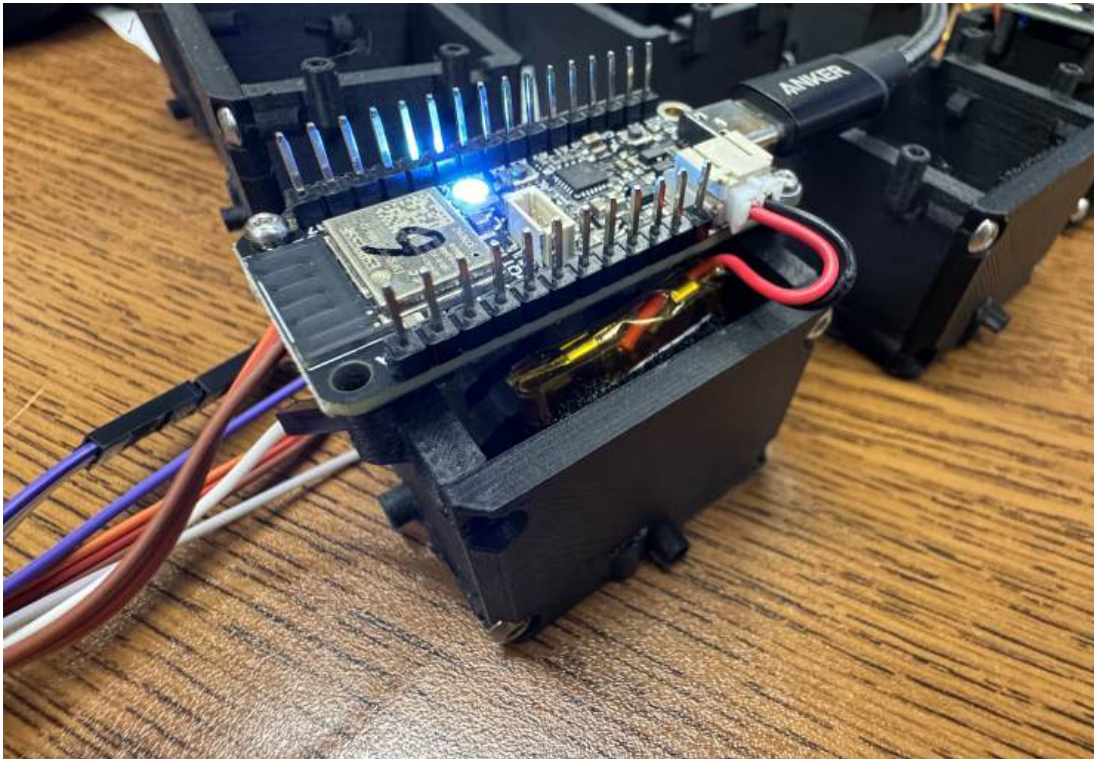


Figure 10: An assembled active node with the Microcontroller and battery connected. Use this orientation for the electronics on each node.

It is quite helpful to use metric hex keys with a handle while assembling a node. Here is an Amazon link to the ones I have been using, in case the set that I bought has been lost in the void that is the CRB makerspace. When mounting the servo,

seat the screws in the holes of the servo first, and then place the entire servo into the node housing and fasten the bolts; trying to insert the screws after you have placed the servo into the housing is quite difficult. Mounting the removable wall and the microcontroller only really requires two screws at opposing corners, but the bistable compliant mechanism requires fasteners at all 4 corners in order to function properly.

The housing and pulley system is designed to be modular and compact. As can be seen in Figure 11, all electrical and mechanical components fit compactly in a 5cm cube. Figure 12 gives a more detailed view of specific mounting points for each component.

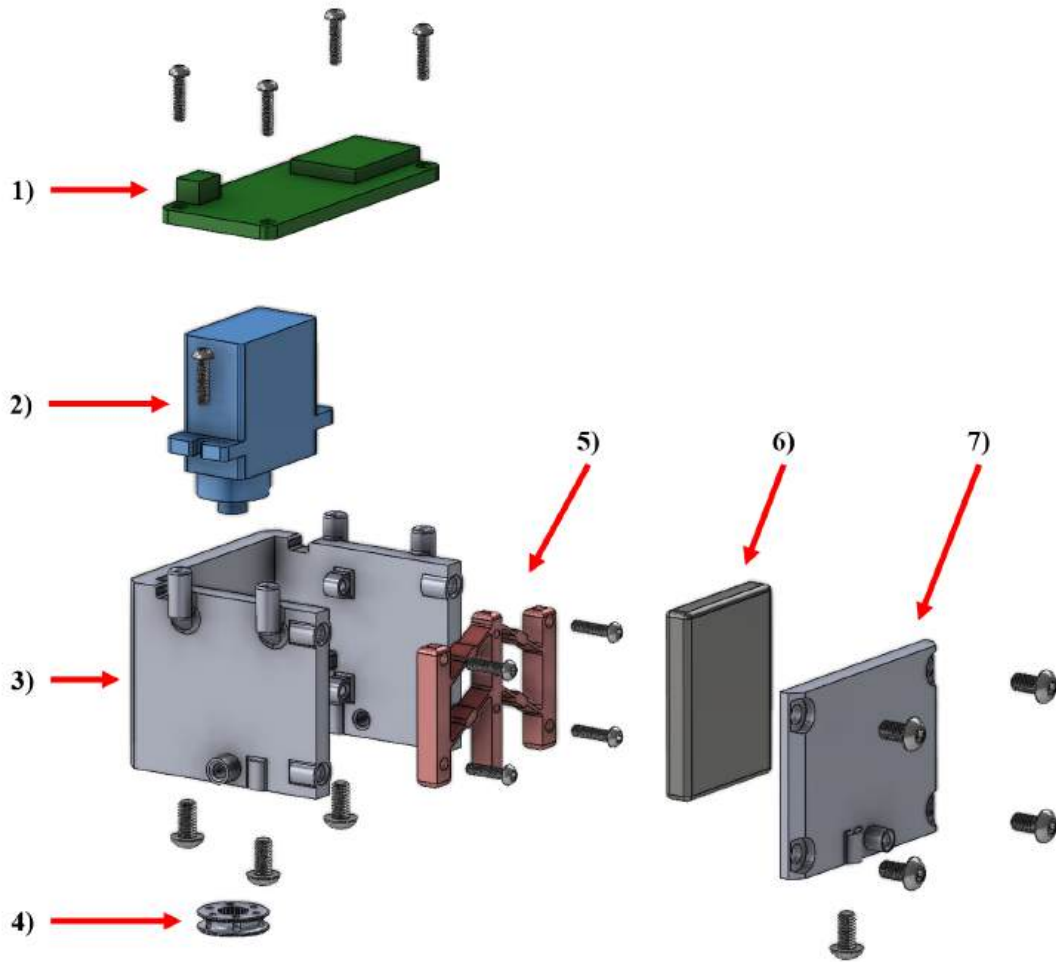


Figure 11: Exploded view of an active node with all components. **1)** Adafruit Feather microcontroller. **2)** FeeTech Servo Motor. **3)** The main body of the node housing serves as the mounting point for all components. **4)** Cable pulley. **5)** Compliant bistable mechanism. **6)** Lithium polymer power supply battery. **7)** Removable lid for assembly and repair.

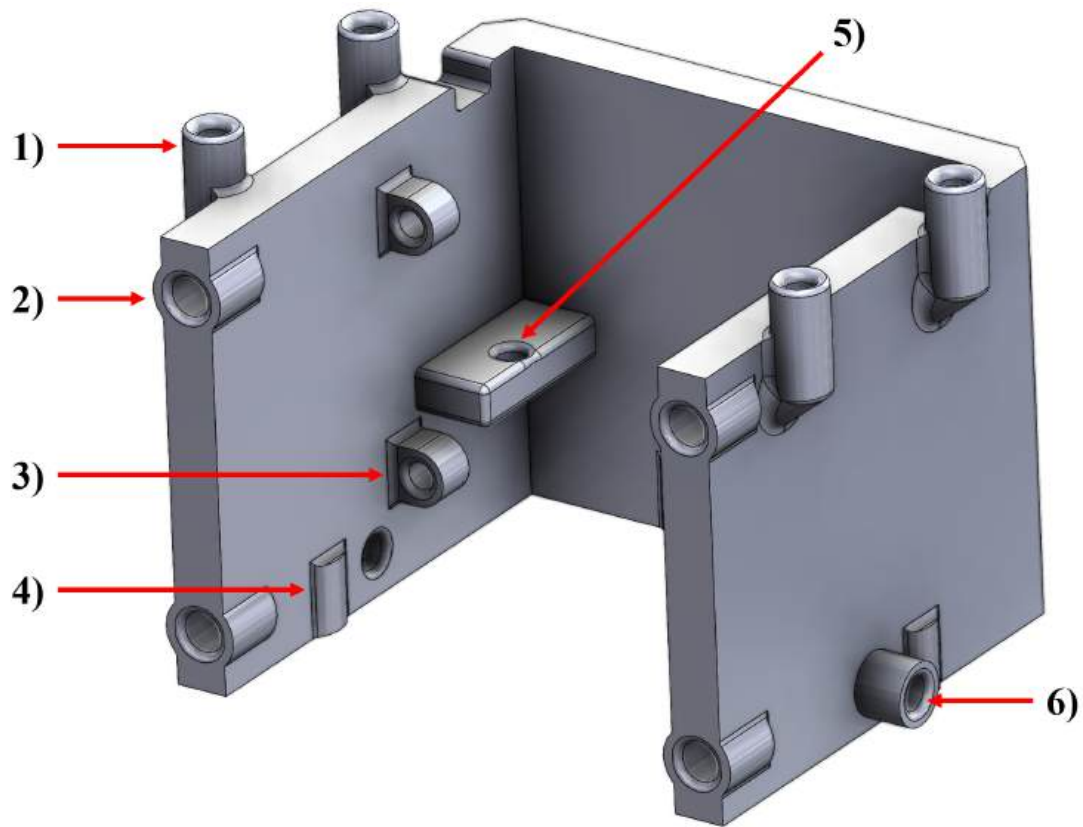


Figure 12: CAD Housing with labeled mounting points. **1)** Mounts for Adafruit Feather microcontroller. **2)** Mounting hold for the removable wall. **3)** Mounting hole for compliant bistable mechanism. **4)** Anchor point for incoming cables. **5)** Mounting point for Servo. **6)** Cable port for incoming and outgoing wire. The cable port also fixes the location of the compression spring.

Step 9: Mount Pulleys & Route Cables

Each servo has an M2.5 machine screw which allows you to mount the pulley to the output gear. First, flip the node system upside down to access the output gear of the servo. Seat, but do not fully tighten, M3 screws in appropriate holes of the housing (item **4**) shown in Figure 12) in preparation for cable routing. Next, wrap each cable around the pulley two to three times and press the top of the pulley onto the servo. With some force, use the M2.5 to mount the pulley to the servo. The amount of torque required to fully seat the screw will cause the servo to rotate, so you must pinch and hold the pulley to fully seat the screw.

Now for the fun part: cable routing! We will discuss inter-node routing first. Using your handy-dandy tweezers, route the fishing line between nodes, through the guide holes and the compression springs. Note that the system should be upside-down throughout this entire process. Figure 13 shows how cables should be routed between nodes.

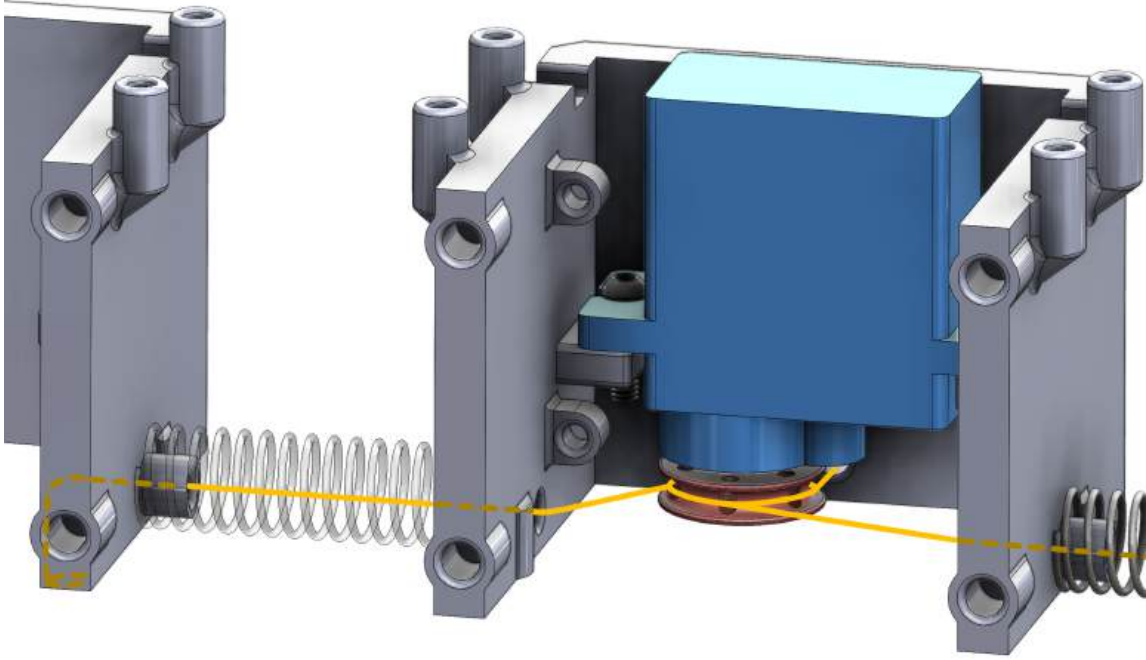


Figure 13: Inter-node routing in metamaterial system.

The cables are terminated at neighboring nodes using an M3 screw, as shown in Figure 14.

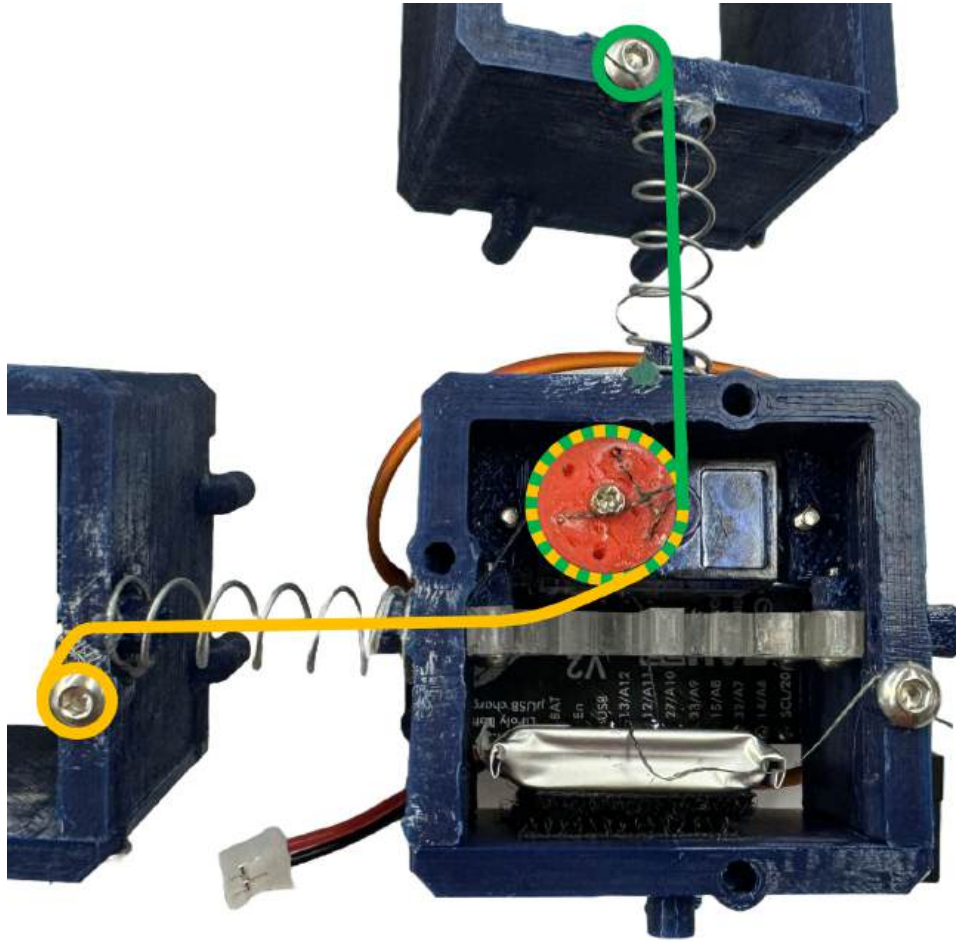


Figure 14: Bottom-view of cable routing between nodes with termination at neighboring passive nodes.

Important note for anchoring cables: When anchoring cables at a neighboring node, wrap them **counterclockwise** around the M3 screw. If you wrap the cable clockwise, it will tense as you tighten the screw into the housing, making it difficult to evenly tension the entire node system. For now, it is perfectly fine to leave slack in the cable system - we will deal with this later.

Now, we will discuss intra-node cable routing. Figure 15 shows two views of how cables are routed to actuate the bistable mechanism.

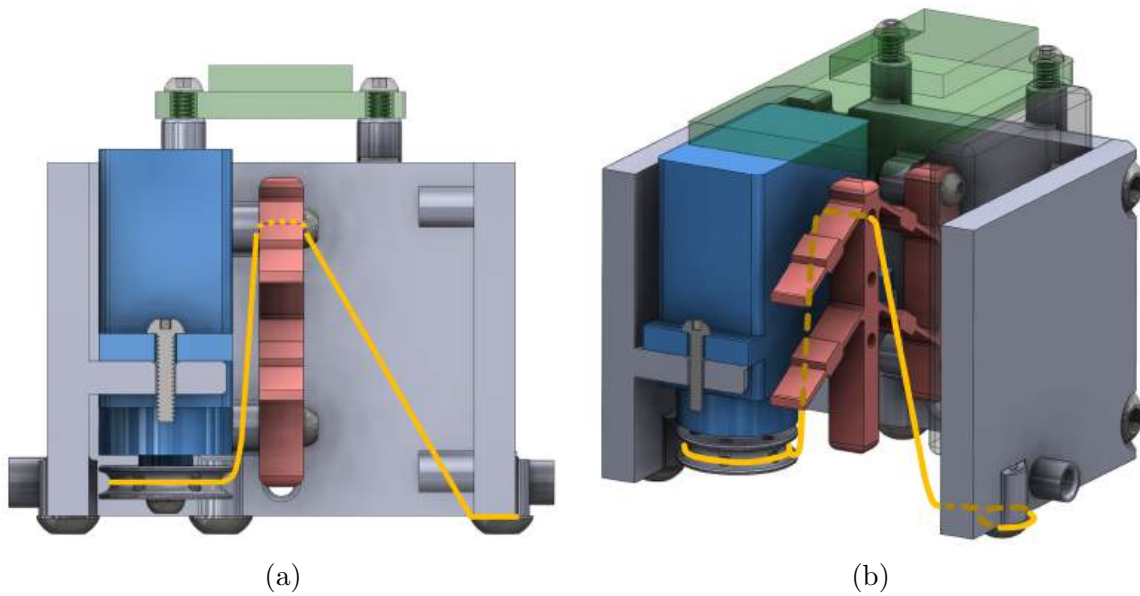


Figure 15: (a) A profile section view of how the bistable mechanism is actuated. (b) An alternate view of how cables are routed through the bistable mechanism.

Routing cables between nodes can be fairly tedious, too. I can typically route the cable through the mechanism while it is mounted to the housing using tweezers, but it is more reliable to just remove the compliant mechanism and route the cable before reattaching it. Similar to the inter-node routing, leave slack in the cable for now. Fasten down all of the anchoring hardware and rejoice! Celebrate having completed such a tedious task!

Step 10: Upload Embedded Firmware & Charge Electronics

Alright, enough revelry. Nobody likes a showboat. Gather your microcontrollers and give them all a distinguishing feature, be that in the form of a sticker, a label, or a number written somewhere on it. This will serve as its address, which will be helpful when controlling the system. When I designate an MCU with an address, I also give its corresponding battery the same address to keep track of charging. Remove the batteries from the node housings and plug them into the microcontroller. Use a USB-C cable to connect the microcontroller to your laptop. Assuming the battery needs charging, you will see a yellow LED illuminate on the board, which indicates

that the battery is charging. I use a USB extender to charge four batteries at once. Charging a battery takes roughly 30 to 40 minutes. Unplug the battery once it is finished charging, and reattach it to the Velcro in the node.

Upload the ‘featherTest.ino’ script to the microcontroller over USB using the Arduino IDE. Repeat this process for each microcontroller in the system. If you aren’t yet able to program the Feather microcontrollers, refer to Section 5.2 before continuing to the next step.

Step 11: Mount & Wire MCU:

Once the appropriate code is loaded onto the Feather, mount it to the top of the node using the M2 screws. Wire the servo to the microcontroller according to Table 5. Repeat this process for each microcontroller in the system.

Servo Wire	MCU Pin
Power (Red)	3V
Ground (Brown)	GND
Control (Orange)	12

Table 5: MCU-Servo Pin connections

Step 12: Test & Adjust Tensioning

This is a good point to test and adjust tensioning for each active node. This is an iterative process that gets easier with practice. Connect a USB-C cable to the MCU and turn the system upside-down. This will allow you to watch the tensioning system when it is active. Run the ‘featherTest.ino’ script provided on the GitHub⁷, which allows you to control a node by giving it integer commands from the Arduino serial monitor. Table 6 lists different commands for the script.

⁷github.com/MaxPatwardhan/multistableRoboticMetamaterial

Serial Monitor Input:	Servo Command
1	CW Slow
2	Stop
3	CCW Slow
5	CW Fast
6	Stop
7	CCW Fast

Table 6: ‘featherTest.ino’ script command chart.

Enter ‘1’ to drive the node slowly. It will be visually apparent which cables need adjustment. At this point, press ‘2’ and subsequently ‘3’, which will drive it in the reverse direction. Once the cables have slackened, stop the rotation and adjust the connections that were quick to tension. Repeat this process at the node until all cables finish contracting at the same time. Then, repeat this process for each active node.

I have found that the cable driving the bistable mechanism should have more slack than the inter-node connections due to a difference in their range of motion. Fully contracting the bistable mechanism requires fewer turns from the servo than fully contracting the compression springs; thus, leaving more slack in the internal cable will cause the mechanism to be driven later in the tensioning process between nodes.

Step 13: Upload MQTT Control Test Code:

With the MCU still connected to your laptop. Upload the ‘mqttTest.ino’ script to test the interface with the MQTT broker. This script will augment the color of the NeoPixel RGB LEDs on the MCU depending on input from the local Python client (no, I am not going to teach you how to install Python on your laptop). Be sure to mirror the physical address you gave to the microcontroller in the embedded code. Note that bestowing the same address to two network clients is **the** optimal way to kill your MQTT network and unintentionally DDoS your local WiFi network.

Once you have uploaded your script to the MCU, unplug the USB-C cable and

the battery. Repeat this process for each MCU in the system.

Step 14: Run the MQTT Broker

Next, we will run the MQTT Broker. Similar to **Step 10**, if you have not set up and configured the MQTT broker, refer to Section 5.1. Otherwise, run the broker on your laptop. Specific instructions for starting an instance of the broker can also be found in Section 5.1.

Once the broker is running, you must ensure it is correctly logging client connections by running the provided ‘mqttTest.py’ Python script and checking that you see the broker log a client interaction.

Step 15: System Test

Connect the battery at each node, and look for each address on the MQTT broker log. The broker utilizes a heartbeat (or Keep-Alive) to ensure all clients are still connected. Figure 16 shows the broker log output for four clients on the MQTT network.

```
Windows PowerShell
1717684831: Sending PINGRESP to esp32-client-3
1717684833: Received PINGREQ from esp32-client-1
1717684833: Sending PINGRESP to esp32-client-1
1717684834: Received PINGREQ from esp32-client-6
1717684834: Sending PINGRESP to esp32-client-6
1717684836: Received PINGREQ from esp32-client-4
1717684836: Sending PINGRESP to esp32-client-4
1717684838: Received PINGREQ from esp32-client-3
1717684838: Sending PINGRESP to esp32-client-3
1717684840: Received PINGREQ from esp32-client-1
1717684840: Sending PINGRESP to esp32-client-1
1717684841: Received PINGREQ from esp32-client-6
1717684841: Sending PINGRESP to esp32-client-6
1717684843: Received PINGREQ from esp32-client-4
1717684843: Sending PINGRESP to esp32-client-4
1717684845: Received PINGREQ from esp32-client-3
1717684845: Sending PINGRESP to esp32-client-3
```

Figure 16: Heartbeat exchange between broker and 4 MCU’s, addressed “esp32-client-#”, on MQTT broker. The heartbeat frequency can be set in the embedded firmware. The heartbeat is a good visual confirmation of all clients on the network.

You can test the MQTT connection with the following inputs to the ‘mqttTest.py’ script:

Cmd Line Prompt:	Action
0	MCU LED Off
1	MCU LED Red
2	MCU LED Green
3	MCU LED Blue
4	MCU LED RGB Strobe
q	Quit Program

Table 7: ‘mqttTest.py’ script command chart.

Assuming everything is working as planned, each microcontroller should change its color based on your input. From here, you can upload any script you desire to each of the boards. Section 5 will discuss how to set up and utilize the aforementioned software.

5 Software & Networking

5.1 MQTT Broker

Message Query Telemetry Transport (MQTT) is an ideal wireless broker-based communication protocol for our application. The broker distributes messages between clients on the network. There are two parts to any message on the MQTT network: the **topic** and the **payload**. This structure allows for precise control over what you can address on the network. A message can take the form of:

$$\text{Message: } \underbrace{\text{nodes/all/servo}}_{\text{Topic}} : \underbrace{\text{'ccw'}}_{\text{Payload}}$$

which would command each node client on the network to drive their servos counterclockwise. Alternatively, a message on the network could take the form:

$$\text{Message: } \underbrace{\text{nodes/4/neopixel}}_{\text{Topic}} : \underbrace{\text{'red'}}_{\text{Payload}},$$

causing the node with address '4' to change its RGB LED to a red color. The overall structure of the network is shown in Figure 17. **Note:** the current system I am employing does not allow you to use Enterprise WiFi networks. It is necessary to use your own router and establish a local WiFi network.

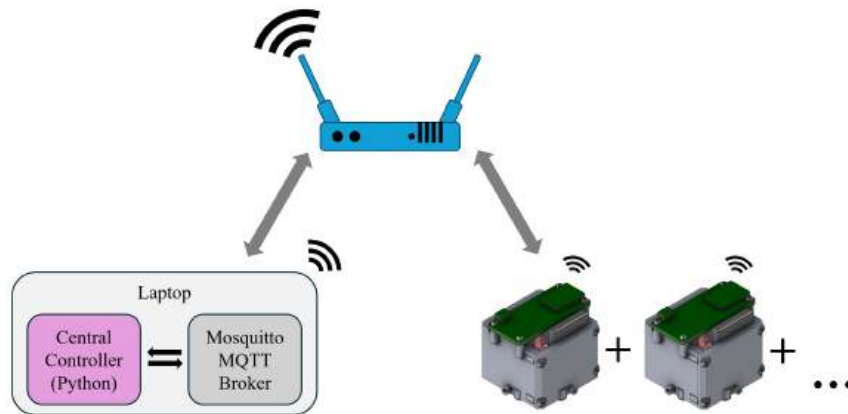
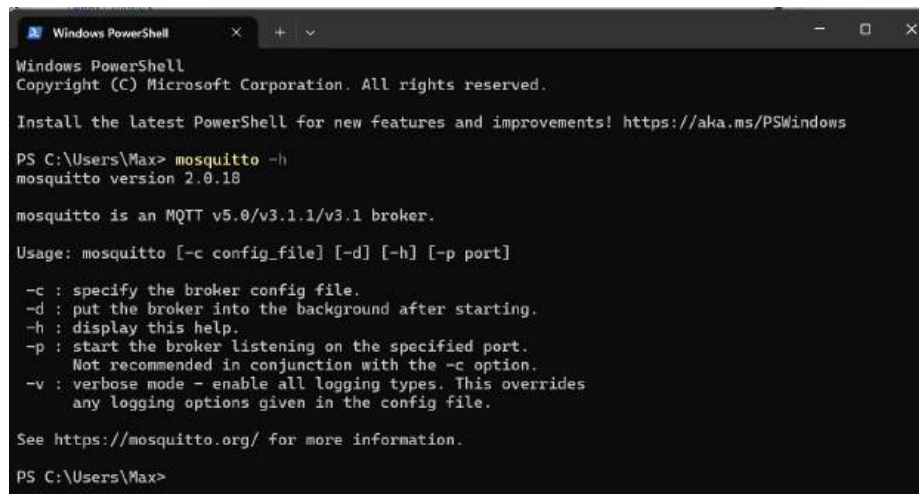


Figure 17: Client-Broker based structure of MQTT.

5.1.1 Installation & Setup

- Step 1: **Download and install Mosquitto** : The download can be found on the Mosquitto website.
- Step 2: **Add Mosquitto to PATH**: This step will allow you to easily launch and utilize Mosquitto from the command prompt. Navigate to 'Control Panel' → 'System and Security' → 'System'. Click on 'Advanced System Settings', and then 'Environment Variables'. In 'System Variables', find 'PATH' and click 'Edit'. Click 'New' and input the Mosquitto installation directory (example: 'C:\Program Files\Mosquitto').
- Step 3: **Verify installation**: Open a command prompt and type the command: **mosquitto -h**.

If you have successfully installed Mosquitto, typing the command in **Step 3** above should output what is shown in Figure 18.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Max> mosquitto -h
mosquitto version 2.0.18

mosquitto is an MQTT v5.0/v3.1.1/v3.1 broker.

Usage: mosquitto [-c config_file] [-d] [-h] [-p port]

-c : specify the broker config file.
-d : put the broker into the background after starting.
-h : display this help.
-p : start the broker listening on the specified port.
    Not recommended in conjunction with the -c option.
-v : verbose mode - enable all logging types. This overrides
    any logging options given in the config file.

See https://mosquitto.org/ for more information.

PS C:\Users\Max>
```

Figure 18: Terminal output of 'mosquitto -h'.

5.1.2 Configuration & Runtime

Mosquitto allows you to input custom configuration files when starting a broker instance. This configuration file allows you to change many properties of the broker,

such as activity logging, IP and Port configuration, and client authentication. The configuration file I use is available on GitHub named ‘mosquitto.CONF’. If you want to use this configuration file, save it within the ‘mosquitto’ directory on your computer. Table 8 lists some of the key configuration settings of my broker.

Setting	Description
log_dest stderr	Sends log messages to stderr which are displayed at runtime
log_type all	Logs all messages including debug, error, warning, notice, information, none, subscribe, unsubscribe, websockets, or all.
listener 1883	Configures port 1883 for the broker to listen for incoming connections.
protocol mqtt	Specifies MQTT as the protocol for the listener, as opposed to WebSockets.
socket_domain ipv4	Forces listener to only use IPv4. Hell will freeze before I use IPv6 for this robot.
allow_anonymous true	Allows anonymous clients to connect.
persistence true	Enables persistent messages, which are stored by the broker and delivered to subscribers not connected at the time of the event.
persistence_location var/lib/mosquitto	Specifies the location for storing persistent messages.

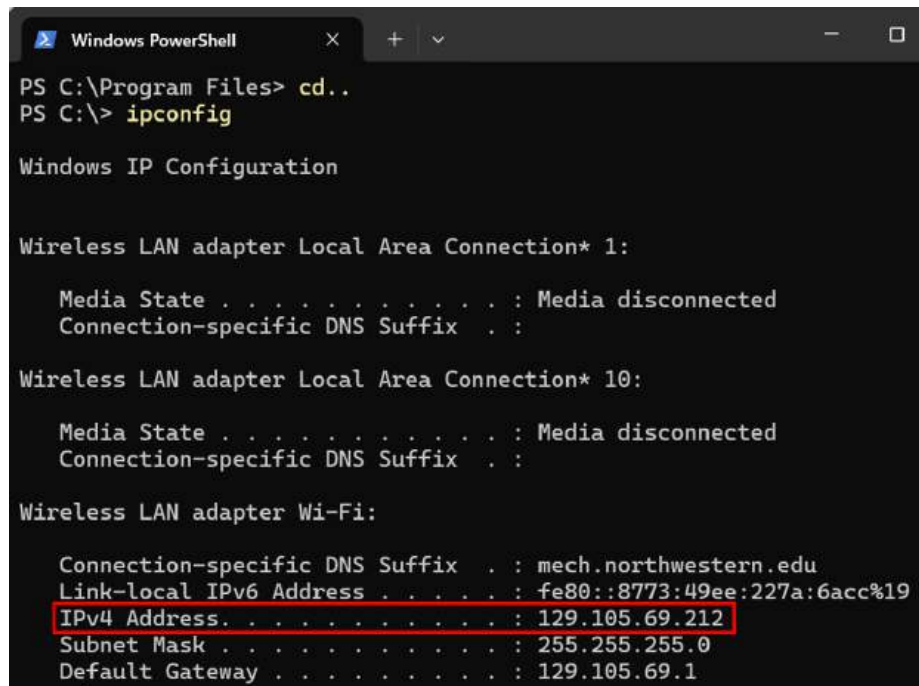
Table 8: Key Mosquitto Configuration Settings

Here are the steps to start an instance of Mosquitto:

- Step 1: **Connect to a local WiFi Network:** “Beware the *eduroam*, my son! The jaws that bite, the claws that catch! Beware the *Device-Northwestern*, and shun the frumious *Guest-Northwestern!*” -Lewis Carroll (Probably)
- Step 2: **Navigate to mosquitto directory:** Open a terminal and redirect to the location of mosquitto (example: ‘**cd 'C://Program Files/mosquitto/'**’).
- Step 3: **Configure Mosquitto:** Start mosquitto with your desired configuration file by entering the command **mosquitto -c 'mosquitto.conf'**.

This will start the broker and log all activity on the terminal.

Now that you have started the broker, you will need to check your computer's IP address in order to program the Nodes as well as the Python client. You can get your computer's IP address by entering **ipconfig** into a terminal. Figure 19 shows where to read your local IPv4 address. This information will be necessary for your embedded code and Python controller, as each client on the network needs to know what address to communicate with.



```
Windows PowerShell
PS C:\Program Files> cd..
PS C:\> ipconfig

Windows IP Configuration

Wireless LAN adapter Local Area Connection* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . : mech.northwestern.edu
    Link-local IPv6 Address . . . . . : fe80::8773:49ee:227a:6acc%19
    IPv4 Address. . . . . : 129.105.69.212
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 129.105.69.1
```

Figure 19: IPv4 address output when using the command **ipconfig**.

You can ensure that the broker is listening on Port 1883 of your laptop by opening a terminal as 'Administrator' and running the command **netstat -an**. This shows the numerical addresses of all active connections and listening ports that your device recognizes. Figure 21 shows a terminal that has run **netstat -an** adjacent to a terminal that has run **ipconfig**. This confirms that the broker is listening on the configured Port.

```
Administrator: Windows PowerShell
PS C:\> netstat -an

Active Connections

Proto Local Address           Foreign Address         State
TCP   0.0.0.0:135              0.0.0.0:*               LISTENING
TCP   0.0.0.0:445              0.0.0.0:*               LISTENING
TCP   0.0.0.0:1337             0.0.0.0:*               LISTENING
TCP   0.0.0.0:1883             0.0.0.0:*               LISTENING
TCP   0.0.0.0:5040             0.0.0.0:*               LISTENING
TCP   0.0.0.0:7680             0.0.0.0:*               LISTENING
TCP   0.0.0.0:49664            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49665            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49666            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49667            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49668            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49669            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49674            0.0.0.0:*               LISTENING
TCP   0.0.0.0:54235            0.0.0.0:*               LISTENING
TCP   0.0.0.0:61218            0.0.0.0:*               LISTENING
TCP   127.0.0.1:5354           0.0.0.0:*               LISTENING
TCP   127.0.0.1:13339          0.0.0.0:*               LISTENING
TCP   127.0.0.1:27182          0.0.0.0:*               LISTENING
TCP   127.0.0.1:49672          0.0.0.0:*               LISTENING
TCP   127.0.0.1:49838          0.0.0.0:*               LISTENING
TCP   127.0.0.1:62522          0.0.0.0:*               LISTENING
TCP   129.105.69.212:1139      0.0.0.0:*               LISTENING
TCP   129.105.69.212:1883      129.105.69.252:57739    ESTABLISHED
TCP   129.105.69.212:54614     3.128.195.20:443        ESTABLISHED
TCP   129.105.69.212:54616     52.11.79.54:443         ESTABLISHED
TCP   129.105.69.212:54624     52.159.126.152:443      ESTABLISHED
TCP   129.105.69.212:54630     34.120.52.64:443        ESTABLISHED
TCP   129.105.69.212:54682     13.107.246.254:443      CLOSE_WAIT
TCP   129.105.69.212:54696     54.209.210.251:443      CLOSE_WAIT
TCP   129.105.69.212:55072     208.95.152.110:443      ESTABLISHED
TCP   129.105.69.212:57004     140.82.113.26:443       ESTABLISHED
TCP   129.105.69.212:57106     34.120.52.64:443        ESTABLISHED
```

Figure 20: Terminal output when using the command `netstat -an` which shows the state of the Port 1883 as “Listening”.

You can now monitor the Log of the MQTT broker, which will display all activity on the network. Figure 1

```
Administrator: Windows PowerShell
PS C:\> netstat -an

Active Connections

Proto Local Address           Foreign Address         State
TCP   0.0.0.0:135              0.0.0.0:*               LISTENING
TCP   0.0.0.0:445              0.0.0.0:*               LISTENING
TCP   0.0.0.0:1337             0.0.0.0:*               LISTENING
TCP   0.0.0.0:1883             0.0.0.0:*               LISTENING
TCP   0.0.0.0:5040             0.0.0.0:*               LISTENING
TCP   0.0.0.0:7680             0.0.0.0:*               LISTENING
TCP   0.0.0.0:49664            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49665            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49666            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49667            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49668            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49669            0.0.0.0:*               LISTENING
TCP   0.0.0.0:49674            0.0.0.0:*               LISTENING
TCP   0.0.0.0:54235            0.0.0.0:*               LISTENING
TCP   0.0.0.0:61218            0.0.0.0:*               LISTENING
TCP   127.0.0.1:5354           0.0.0.0:*               LISTENING
TCP   127.0.0.1:13339          0.0.0.0:*               LISTENING
TCP   127.0.0.1:27182          0.0.0.0:*               LISTENING
TCP   127.0.0.1:49672          0.0.0.0:*               LISTENING
TCP   127.0.0.1:49838          0.0.0.0:*               LISTENING
TCP   127.0.0.1:62522          0.0.0.0:*               LISTENING
TCP   129.105.69.212:1139      0.0.0.0:*               LISTENING
TCP   129.105.69.212:1883      129.105.69.252:57739    ESTABLISHED
TCP   129.105.69.212:54614     3.128.195.20:443        ESTABLISHED
TCP   129.105.69.212:54616     52.11.79.54:443         ESTABLISHED
TCP   129.105.69.212:54624     52.159.126.152:443      ESTABLISHED
TCP   129.105.69.212:54630     34.120.52.64:443        ESTABLISHED
TCP   129.105.69.212:54682     13.107.246.254:443      CLOSE_WAIT
TCP   129.105.69.212:54696     54.209.210.251:443      CLOSE_WAIT
TCP   129.105.69.212:55072     208.95.152.110:443      ESTABLISHED
TCP   129.105.69.212:57004     140.82.113.26:443       ESTABLISHED
TCP   129.105.69.212:57106     34.120.52.64:443        ESTABLISHED
```

Figure 21: Terminal output when using the command `netstat -an` which shows the state of the Port 1883 as “Listening”.

5.2 Embedded Software

5.2.1 Installation & Setup

The Adafruit Feather ESP32 V2 Microcontroller platform is programmed using the Arduino IDE using a USB-C cable. Here are the steps to set up the IDE for programming this board:

- Step 1: **Download the Arduino IDE:** This can be found on the Arduino website.
- Step 2: **Ensure you have the correct drivers:** This shouldn't be an issue if you are using a Mac or Linux OS, but sometimes Windows will be missing drivers for the USB to UART bridge on the board. They can be downloaded from the Silicon Labs CP210x website.
- Step 3: **Add the ESP32 package to the Arduino Board manager:** In the Arduino IDE, go to 'File' → 'Preferences'. There, you will find a field named "Additional Board Manager URL's". Paste this URL:
`https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
- Step 4: **Install ESP32 Board Support:** Go to 'Tools' → 'Board' → 'Boards Manager'. Search for 'esp32' and select 'Adafruit Feather ESP32 V2'. Now, navigate to 'Tools' → 'Board', and select 'Adafruit Feather ESP32 V2'.
- Step 5: **Install requisite libraries:** Go to 'Sketch' → 'Include Library' → 'Manage Libraries', and add the following libraries: **'WiFi'**, **'PubSubClient'**, and **'Adafruit Neopixel'**.
- Step 6: **Connect the MCU:** Use a USB-C Cable to connect the microcontroller to your laptop. Go to Tools → Port, and select the appropriate port.

From here, you can try uploading a simple sketch to make sure the system is functioning properly. A good test script is given below:

```
1 const int ledPin = 13;
2 void setup() {
3   pinMode(ledPin, OUTPUT);
4 }
5 void loop() {
6   digitalWrite(ledPin, HIGH); // turn the LED on
7   delay(1000);                // wait for a second
8   digitalWrite(ledPin, LOW);  // turn the LED off
9   delay(1000);
10 }
```

5.2.2 Connection over Wifi

Hello detective/recruiter, this part is still under construction and will be uploaded soon!

6 System Deployment:

System Deployment Process

- Step 1: **Start Mosquitto Broker:** The broker will run locally on your laptop and will require a local WiFi network. Details on how to get this system working are provided in Section ??.
- Step 2: **Connect Batteries to Microcontroller:** Power on each node by connecting the battery.
- Step 3: **Connect Python Client to Broker:** Run the Python controller script on your laptop.
- Step 4: **Ensure and Test Connection:** In a separate terminal, check the message log of the broker to ensure that each node is connected. Every client on the network should send a periodic heartbeat, which will be displayed in the message log.
- Step 5: **Control the Robot!** (I hope it does something cool)
- Step 6: **System shutdown:** When finished using, unplug or charge batteries.

7 Notes

7.1 Hardware

All necessary hardware is listed in the Bill of Materials, Section 2. Importantly, the vendor and part number are listed in this table, so specific details can be referenced on the vendor website. I will discuss characteristics I found to be important when specifying hardware in this section.

The system uses M3 and M2 button-headed cap screws sourced from McMaster-Carr. These bolts are driven with metric hex keys, though their size is somewhat nonstandard due to the smaller dimensions of the bolts. The M3 machine screws are driven by a 2mm hex key, and the M2 machine screws are driven by a 1.3mm hex key. The compression springs are one inch long, with a 0.3-inch outer diameter. They are flattened on the ends and have a spring constant of $k = 3 \frac{\text{lbs}}{\text{in}}$. I found these dimensions to best exhibit the spring-bifurcated behavior in the robot. I have not explored using stiffer springs - I believe you could experiment with a spring rate of $k \approx 5 \frac{\text{lbs}}{\text{in}}$, but anything greater might be too much for the servo motors.

I found that the socket for the M2 screws is somewhat prone to stripping, maybe 10% of the time, which is just frequent enough to be annoying. I might recommend switching to a slightly stronger metal in future iterations. Also, for some reason, I have struggled to find many hex keys that drive the M2 screws. I would recommend switching to a Phillips or Flathead screw. I have had no issues with the M3 screws.

7.2 Bistable Mechanism

There is not much to note regarding this specific mechanism. Its stiffness is tuned so that it may be driven by the servo without stalling it. The design for the mechanism was inspired by Jensen et. al. [1], though specific dimensions for the arm structure were determined through repeated iteration. I believe there exist many opportunities for future research with this mechanism as a pure mechanical sensor or as a mechanical sensor coupled to electronic feedback sensors. One idea is to optimize this mechanism for locomotion and grasping tasks. I would try to achieve this by lengthening the

central piston and adding some sort of ‘foot’ to the bottom of it, allowing for greater surface area and, thus, greater friction with the ground.

References

- [1] *Design Optimization of a Fully-Compliant Bistable Micro-Mechanism*, vol. Micro-Electro-Mechanical Systems (MEMS) of *ASME International Mechanical Engineering Congress and Exposition*, 11 2001.