Autonomous Heat-Seeking Turret

Alex Niculescu & Jamie Wang

ECE 590: Full-Stack IOT Systems

4/26/2024

**Introduction and Motivation**

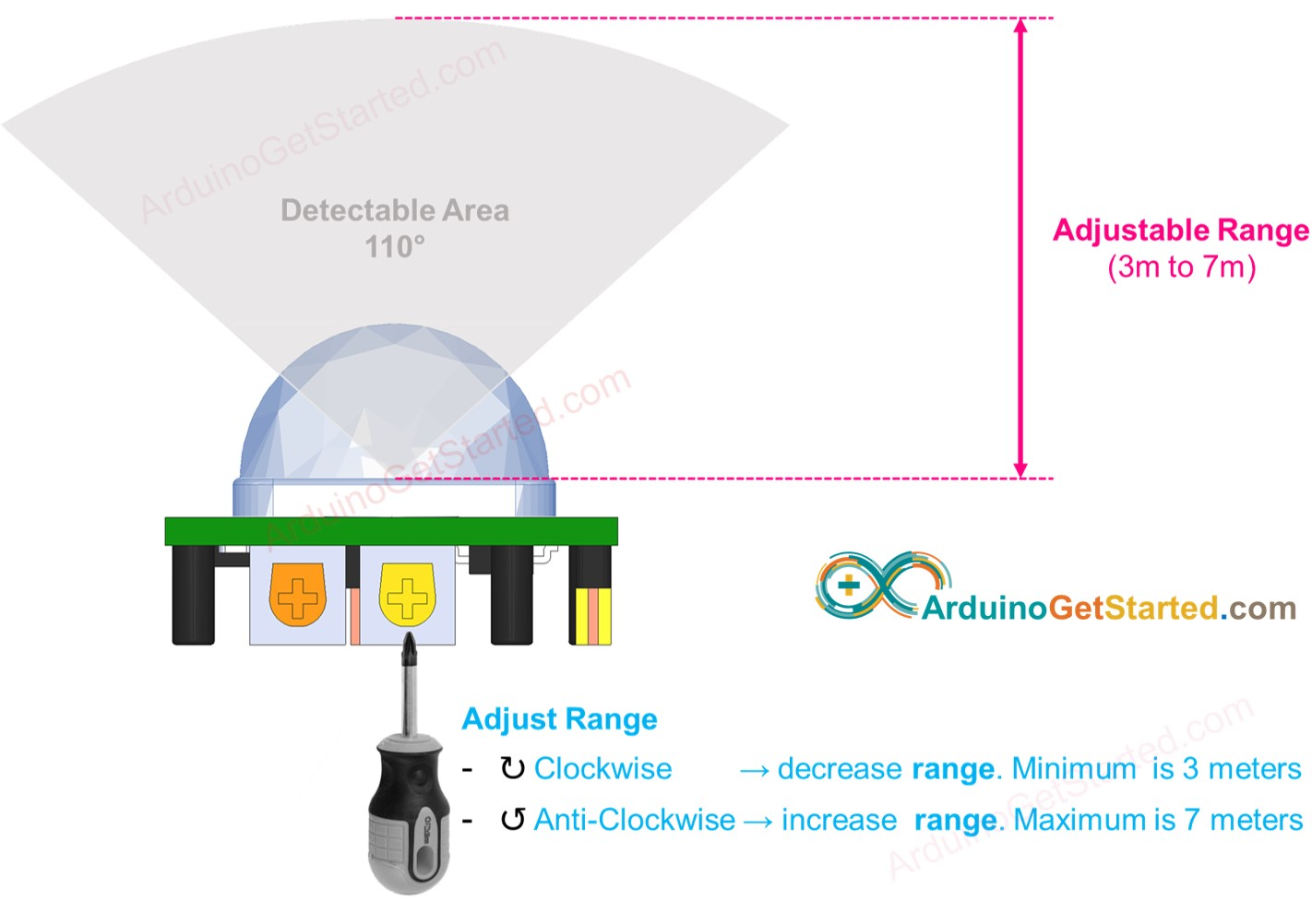
Our group decided to create an autonomous turret that targets moving objects with heat signatures and automatically fires at it. We believe this project uses a lot of the concepts learned in class such as reading data from sensors and using HTTP servers to communicate between devices. The main motivation behind this project was to make a fun product anyone could use. While there are many devices and products that are autonomous, there are not that many toys that even children could use. There are also not many products that can be used in multiple different ways like ours. Our product can be used both manually and autonomously, and can have different applications for both. For example, in autonomous mode, the turret could be used to passively defend against others, or as a toy for oneself to run around and see if it can be tricked or dodged.

Our original timeline was to have the physical device ready to be tested by the second week of April, after which more time would be spent on the software and making the mobile app, leaving a week for testing and refinement (See Appendix A). However, due to unforeseen delays in shipping and acquiring parts, we had to switch plans and start the software first and then integrate it with the physical design. This created some problems later between how the software was initially designed and how the physical system was eventually created, but ultimately, we were able to solve many problems and create a workable turret.

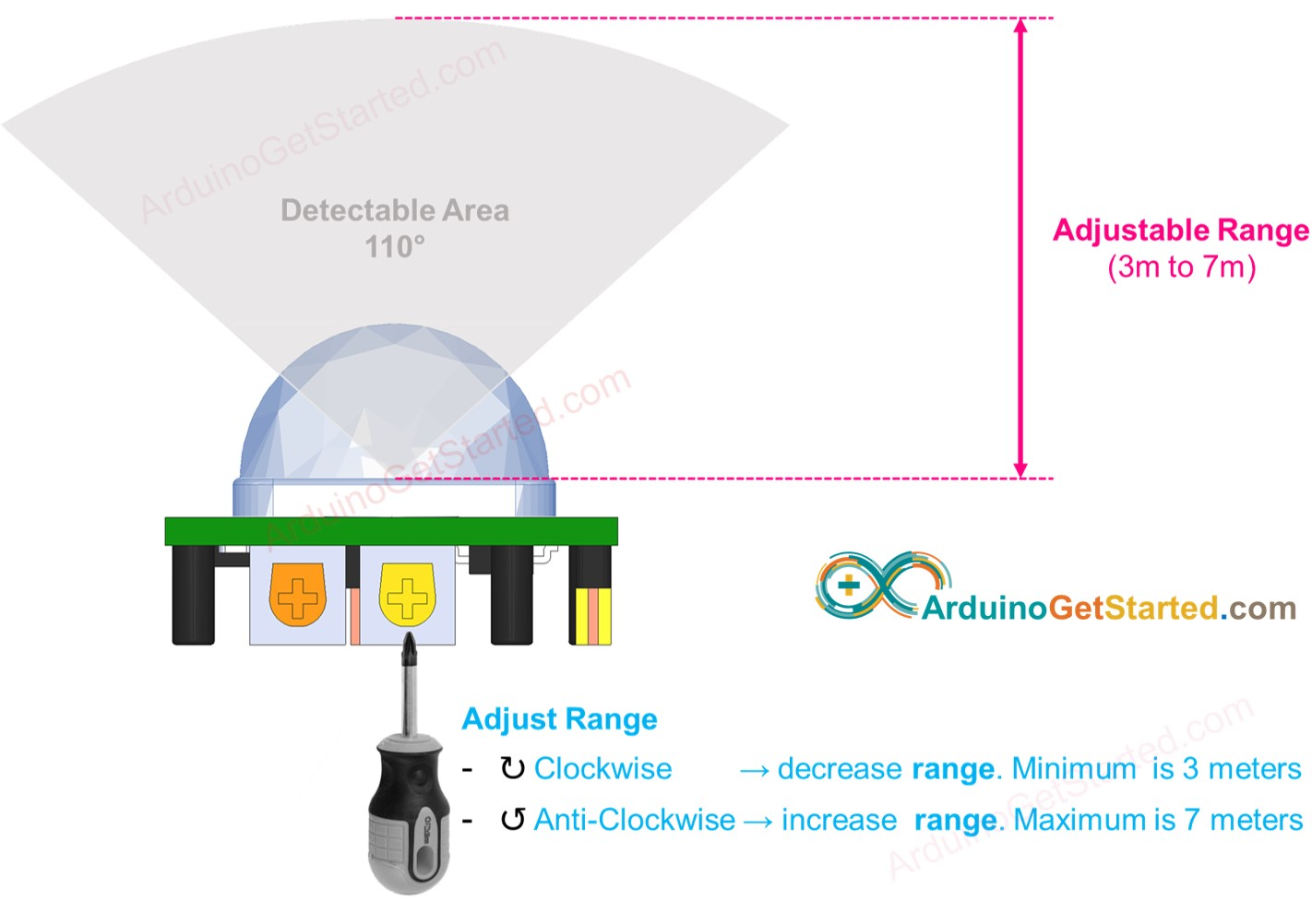
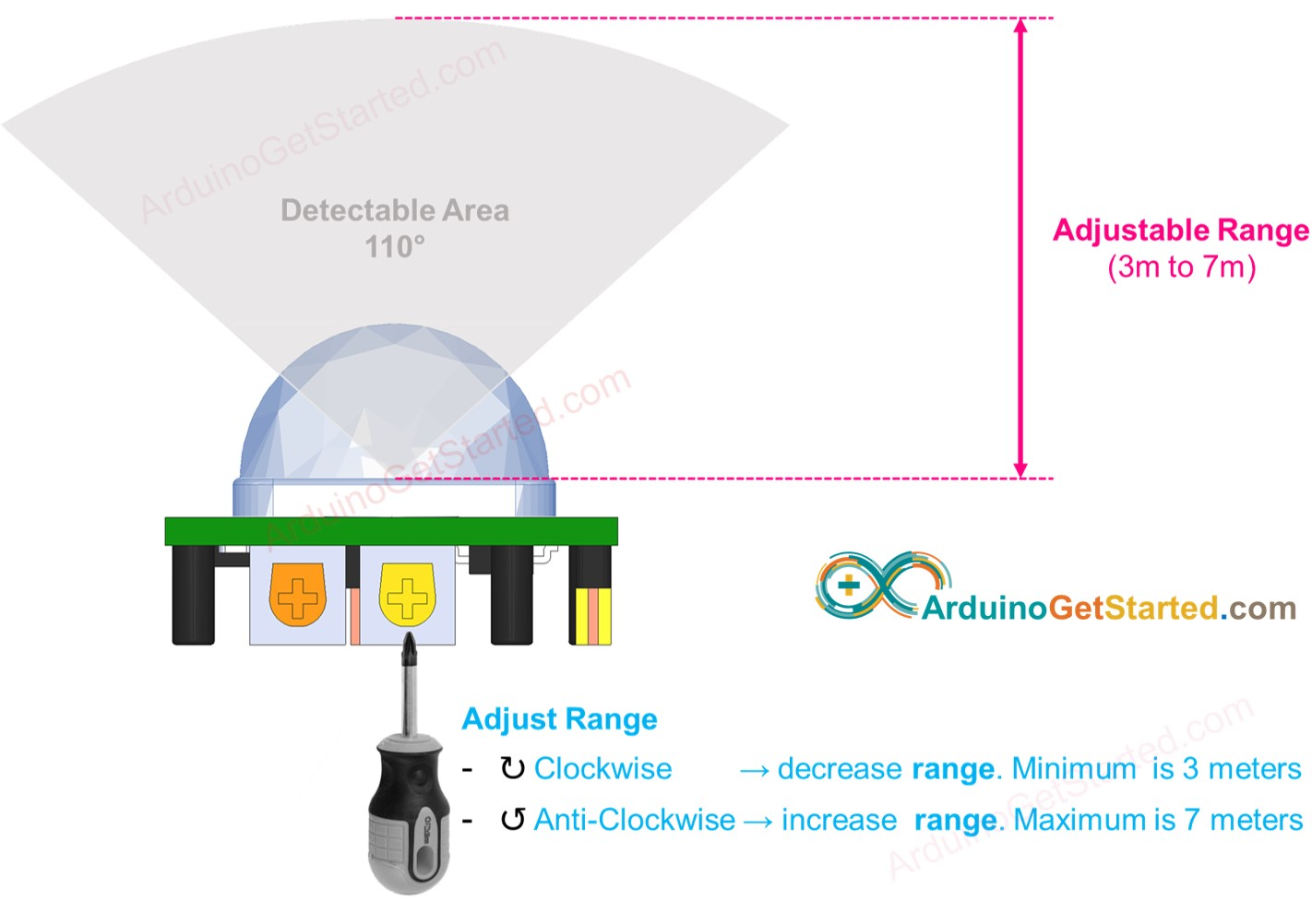
**System Design**

The overall design of the system includes two parts, the physical turret, and the mobile application and HTTP server. The HTTP server which controls the turret receives commands from the mobile app through HTTP POST and GET requests. These commands include: turn left, turn right, fire, and switch to autonomous or manual mode. We also wanted to create a camera feed that would stream to an RTSP server that the mobile app could connect to, allowing the user to see what the turret is pointing at without being nearby. For the actual movement of the turret, we used a DC motor to rotate the gun on two axes. We decided to not try using the third axis of up/down as that would significantly increase the difficulty of both the movement and it would be extremely difficult to measure the height of a detected object. We also needed another motor to trigger the gun to start firing.

For detecting moving objects, we wanted to aim for living targets, so we required sensors that only detect heat and can amplify heat signatures to actually differentiate them from the ambient heat. PIR sensors are exactly applicable for this kind of application, so we wanted to make a ring of PIR sensors around the turret to more accurately detect the motion of an object (See Figure 1).







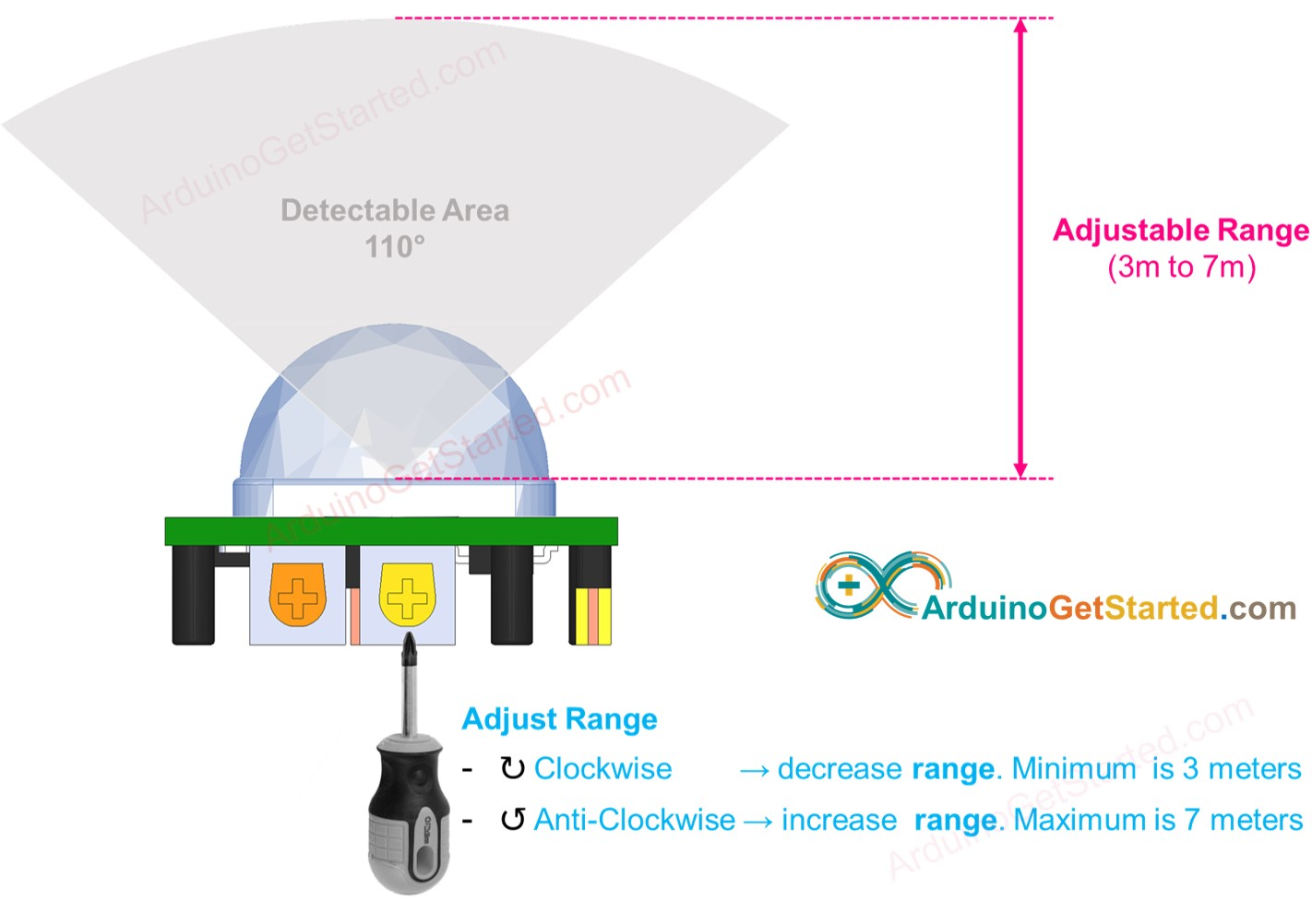


Figure 1. Ring of PIR sensors

For the software side, we wanted to design a system that would take in the inputs of the app and the sensors, and create outputs to the motors. To know how much to spin the motors, we also needed encoders attached to them to measure their speed and location. Using the encoders, we could create a PID (Proportional Integral Derivative) system to control the speed of the motors so they would point to the detected object and also not overshoot its destination. We also wanted to attach a camera to the top of the gun

**System Implementation**

We split our design into three parts: mechanical, electrical, and software. Although a majority of the work went into creating the software, the mechanical and electrical components had to be designed early so that there was enough time to work on the software. It was difficult to create specific designs before we had the Ragefire gun in hand since we were not able to find any sort of dimensions or thorough descriptions of all of its working parts.

**Mechanical -**

Due to the large amount of movement in our project, special considerations had to be taken into account about what materials and physical designs would be needed. The gun weighs approximately 5 lbs, is around 3 feet long, and needed to be rotated at a reasonably fast pace to keep up with a moving target. These requirements required us to have to buy a sizable DC motor, so we bought a motor with a 43.7:1 metal gearbox that could provide 18 Km/cm of torque. Also, the motor has an encoder included with it which is critical for when software development started.

Additionally, the trigger for the gun is atypical in that it not only needs to be pushed in to start, but also needs to be pushed downwards with enough force to start the rotation of the barrel. Even for a human it required us to use both of our thumbs to keep the trigger down, meaning we also needed a powerful motor to actuate the trigger. For this purpose, we bought a high-torque, 20kg servo motor that could be easily controlled and rotated for precise movement.

Finally, we also required ways to mount these motors securely and be connected to their respective components. The DC motor used to rotate the gun was simple to mount, as we were able to replace the rotation coupling that came with the Ragefire gun with our own 3D printed designed part (Appendix B, Figure 3). While it took a couple of iterations to decide on the best design, our final design had the same mounting design as the original coupling, making use of the system that was already in place for the gun. Unfortunately, this created problems later with wiring which will be discussed in the electrical section. The second motor mount we needed was for the servo motor. It needed to be mounted behind the gun where the trigger was but also needed a way to push down on it. We designed another 3D printed part, press-fit the motor into it and secured the part to the handle right behind the trigger. We then attached a servo horn such that it would press on the trigger from above, and since the servo was so close to the trigger, it would also push it forward, firing any loaded bullets (Appendix B, Figure 4).

**Electrical -**

Implementing the electrical design was more complicated than the mechanical side. First, we needed to decide on the microcontroller to control the system. We originally started using the Heltec WiFi LoRa 32 but then realized that its limited number of pins, small memory, and unstable internet connection would hinder our development. For these reasons, we decided to switch to using a Raspberry Pi 4 which had more pins for camera integration, 8 GB of RAM (compared to the Heltec’s 512 KB), and was able to maintain a stable connection to the internet without breaking the rest of the software. Another consideration we had to take into account was the power sources required to power the system. Since we were using a beefy motor to rotate our turret, we also needed a DC motor driver that could handle the power requirements needed. We bought a 3A motor driver to drive the motor, but it needed 12 V to drive it so we also bought 12V batteries. However, the Raspberry Pi could not be powered by 12V, so we needed to also buy a DC buck converter to step-down the 12V to 5V to power both the Raspberry Pi and the servo motor (which was rated for a voltage between 4.8V and 6.8V).

We had originally planned to use around 20 PIR sensors for more accurate position detection, but we did not have the time or resources to make a large enough ring or to create good enough shrouds around the sensors. Since they have a 90° field of vision, putting multiple sensors on the same plane is ineffective since they will detect the same objects in a large area. To counteract this, we made shrouds from tape which limited their field of view so there was less overlap, making calculation of the target position more accurate. However, with 20 sensors, we would have to create much bigger and denser shrouds which we were not able to do. Because of these problems, we started with only using 6 sensors and planned to increase the number later. However, we were able to get decent results with the 6 sensor setup so we did not increase their number in case we would have to go through the process of threshold calibration again (detailed more in the software section).To simplify mounting, physically made a ring of cardboard that we taped to the legs of the tripod and cut holes into it for the sensors to stay in. The bulb on the sensors stopped it from falling out and the small hole still allowed us to easily connect wires to the sensor pins.

To mount the rest of the electronics, we ziptied the Raspberry Pi as well as the breadboard and buck convertor to one of the tripod legs so the DC motor and sensors could easily be wired up to it. However, a problem with this setup was the threat of the servo wires getting tangled and wrapped around the turret due to it being mounted on the rotating gun, while the other electronics are mounted on the unmoving tripod. There are solutions to problems like this, one of them being a slip ring, but unfortunately we could not buy a slip ring before knowing the dimensions of the turret, but by that point we received the turret, it was too late to buy more supplies. To counteract this, we cut long lengths of extra wire and soldered them to the servo wires to extend their length. This way, the turret will have lots of wire to pull from in the event it wants to rotate many times in one direction. The only downside is that there is a limit to how many times the turret can rotate, but in a realistic scenario, the turret would turn in both directions to get to different points (unless the target purposefully just runs in one direction for many rotations).

**Software -**

The software script on the Raspberry Pi makes extensive use of threads to control multiple objects. This is because the server needs to always be running and receiving requests and cannot freeze to carry out another function. So there must always be a thread processing any HTTP requests.

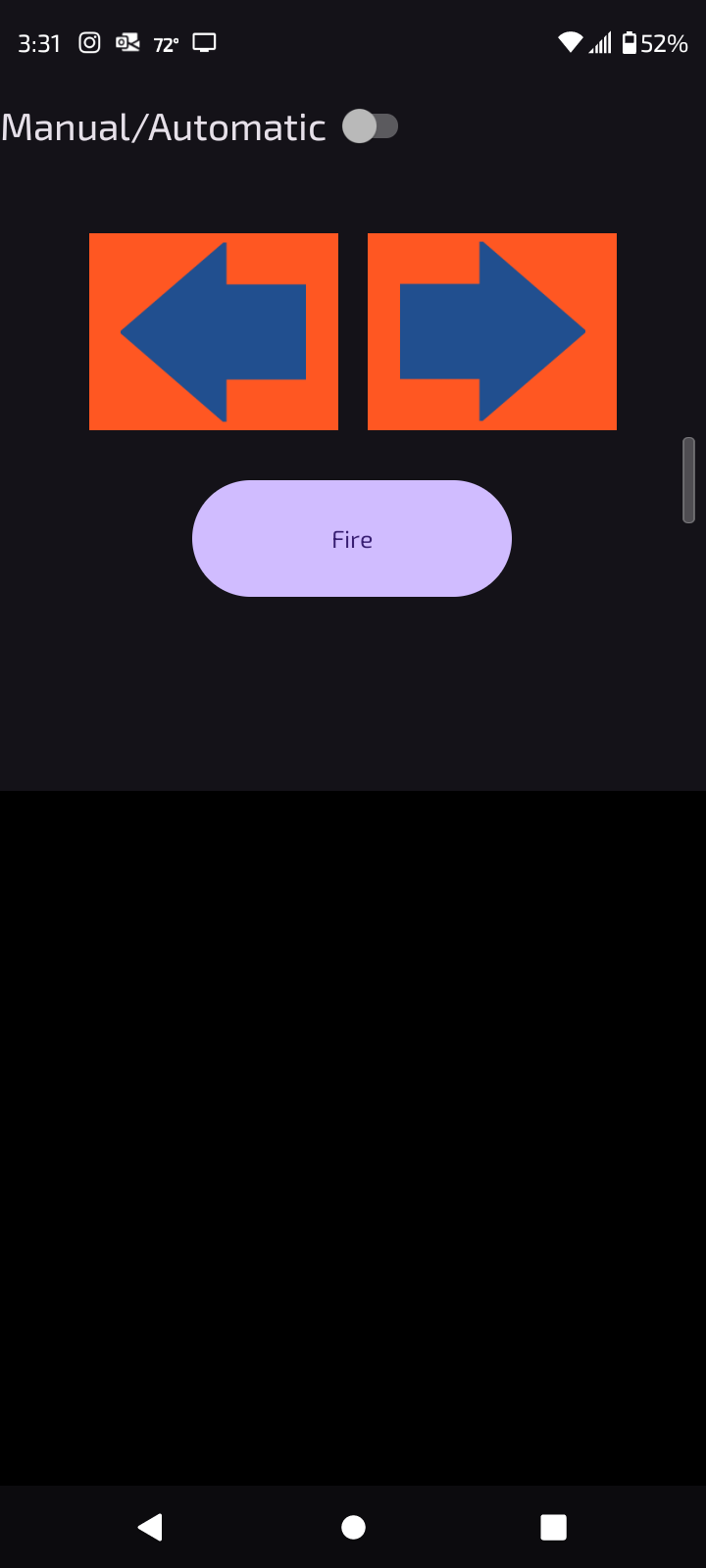


Figure 2. Mobile App

First, to understand the manual mode, the requests that the mobile app sends must be understood first. There are two events that the mobile app listens for, when a button is pressed, and when it is depressed. These listeners help when the user holds down a button to turn a motor for a longer time, rather than having to press the button multiple times. When a button is pressed, a POST request is sent to the server with the “start” command of that particular function (for example, if the “Fire” button is pressed, then “Start fire” is sent to the server). When the user lets go of the button, another POST request is sent with the “stop” command. The only exception to this rule is the switch to turn the turret into manual or autonomous mode, there is no listener for when the button is held, only when it is clicked. Finally, the server will respond with a short text response telling the HTTP client that it successfully received the request and if any error occurred in completing the request (like trying to turn the motor when the turret is in autonomous mode).

With this functionality in mind, the server can be simplified while in manual control, as the server thread can easily carry out movement functions like turn the motor or fire the turret. These functions only require one line of code (turn on or off a pin) so a separate thread is not needed for manual mode. However, when the user wants to set the turret into autonomous mode, there must be another thread that runs that constantly reads the sensor outputs and turns the motors accordingly.

For autonomous functionality, the separate thread goes through a loop that checks the state of each PIR sensor, sums up the positions that each sensor is connected to, and takes the average. For example, if sensors 1, 2, and 3 correspond to 60°, 120°, and 180° respectively, and sensors 1 and 2 detect motion, then the position the turret would turn to would be the average of the two numbers, or 90°. However, with this method comes a suite of problems including tracking multiple objects, sensor delays, and index wrapping. Also, there is a third thread that is fired once the turret reaches its target destination whose sole purpose is to turn the servo, trigger the gun and firing on the target.

**Multiple Objects:** To solve the problem of when there are multiple objects being sensed, our algorithm only checks sensors in sets of 4, starting with sensor 1[[1]](#footnote-0). Once the algorithm finds a sensor that senses an object, it will check up to the next 4 sensors to see if they are detecting as well. If one of the sensors is not detecting, the algorithm will stop analyzing the sensor output and calculate the target position.

**Index Wrapping:** The sensors are mounted in a circular shape, but the list structure we use to link a sensor to a position is linear and does not wrap around. This presents a problem in the case where sensors 1 and 6 detect the same object since they are physically next to each other. The algorithm described above would see that sensor 1 has a high output, and then look at sensors 2, 3, and 4 to see if they are high, without ever looking at sensor 6. To solve this problem, we included certain edge cases where sensors 5 and 6 are checked first if sensor 1 is high before checking any other sensors to finish creating the set of 4 sensors.

**Sensor Delays:** One peculiarity we discovered with the PIR sensors is that once they detect motion, they output a high signal for around 6 seconds afterwards. We did not have a way to adjust this in time without intricate soldering work, but we also could not sample our sensors every 6 seconds as that would introduce a large delay into our system and make it very impractical to use. So instead, we introduced another algorithm that checks if a sensor signal has been high for a period of time, and internally sets that signal to low if so. The internal signal will not be set to high until the requisite time has passed, at which the signal will go high again if the sensor output is still high (meaning the object is still at that location). While this is an imperfect method, we were able to get decent results with it after a heavy amount of threshold calibration. The libraries we used to control the sensors also have different ways to control the sensitivity of the sensors which we also used to try to counteract the issue of the delays. Many variables like how long to wait to turn an internal signal back to high or when to start moving after a high signal was detected had to be tested and changed multiple times to get the best results.

**Experimentation and Evaluation**

Much of the testing and experimentation was done in the Co-lab Studio with the turret standing on top of a table. This was done for safety reasons so that no one would get hit by the rotating gun. The manual controls were quite easy to test and did not present any major problems. However, the autonomous tracking algorithm created lots of problems which took up most of the testing time, most of them stemming from the PIR sensors. The original algorithm did not take into account the delays in the sensor outputs, which would create a lot of confusion in the algorithm, making it believe an object that had already passed was still there long after it was gone. Another problem is that the sensors are heavily affected by light and can easily mistake changes in lighting for motion. While many of these issues were solved with changing the algorithm and using different parameters when interacting with the sensor software libraries, there are still some bugs and false positives that occur when using the turret.

Through our experiments and testing, we were able to find a good balance when changing the timing threshold parameters as noted in the previous section. One thing we discovered, however, is that these parameters have to change when in different lighting conditions. As most of our testing was done in the Co-lab, we found good values to use there, but when we moved to a lab space to test, the tracking quality had decreased but slightly improved when we changed the threshold values again.

In the end, we were able to create a product that can track a moving target with only a couple seconds of delay and accurately fire at it. Since the turret is almost always calculating a new target due to sensor delays and false positives, it does trigger the firing servo before it points at the actual moving object, but with enough ammo, this does not present much of a problem (other than cleaning up afterwards).

**Conclusions and Future Work**

We believe we have learned a lot through this project. We were able to practice many of the embedded systems concepts we learned through this course like using GPIO, timers, actuators as well as networking like HTTP servers and server-client architecture throughout this project. While we did not achieve all of our minor goals, we believe that we created all of the major components and successfully completed the spirit of the project.

Given more time, we would have used more PIR sensors and made a higher fidelity sensor ring and shrouds to have more accurate readings. This would have also decreased the number of false positives the PIR sensors sensed. With more resources, we may have used a different sensor or used the PIR sensors in conjunction with another sensor like an ultrasonic sensor to act as a second level of detection to remove any false positives. Another option would have been to create our own PIR sensors that had the exact behavior we wanted to fit this project, but that would have taken a lot more money and time than we could afford.

Another area of improvement would be to add more internet functionality. One of our low-priority goals was to add a camera to the turret so the user could receive a live feed through the mobile app. Unfortunately, we were not able to implement this feature due to constraints from the rotation of the turret and the fragility of the PiCamera we acquired to interface with the Raspberry Pi. Furthermore, we could have added more features to make it more like a security device. While our purpose for making the turret was just for games, it has genuine capability for other applications. It could act as a security system that tracks movement (although without firing on it) and records the activity. An improvement could be to add connection to a server that logs these movements so that the user can look back and see what moved nearby, like a home security camera.

While our project was a fun toy that anybody could use, it uses many IOT concepts that are used in common devices today. With some transformation, our product could be used for a whole host of other applications. We succeeded in taking the concepts we learned in class and applying them to a workable and engaging product.

Appendix A - Initial Schedule

| Week 3/18 | Finish ordering parts |
| --- | --- |
| Week 3/25 | Test and assemble individual components |
| Week 4/1 | Couple components Create control flow |
| Week 4/8 | Finish mobile app |
| Week 4/15 | Testing and Refinement |
| Week 4/23 | Presentation! |

Appendix B - Project Images

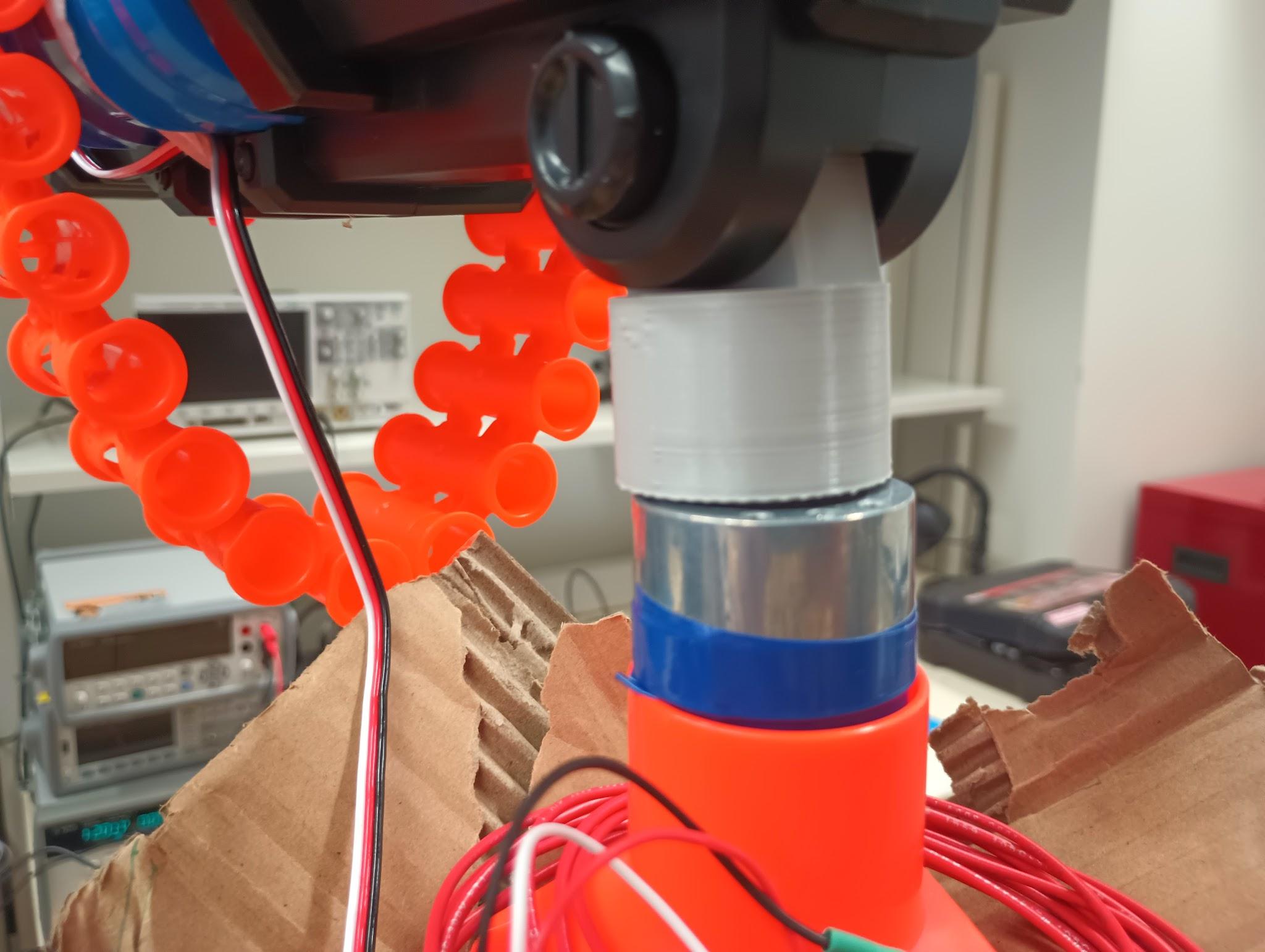


Figure 3. DC Motor mounted in tripod and coupled with gun

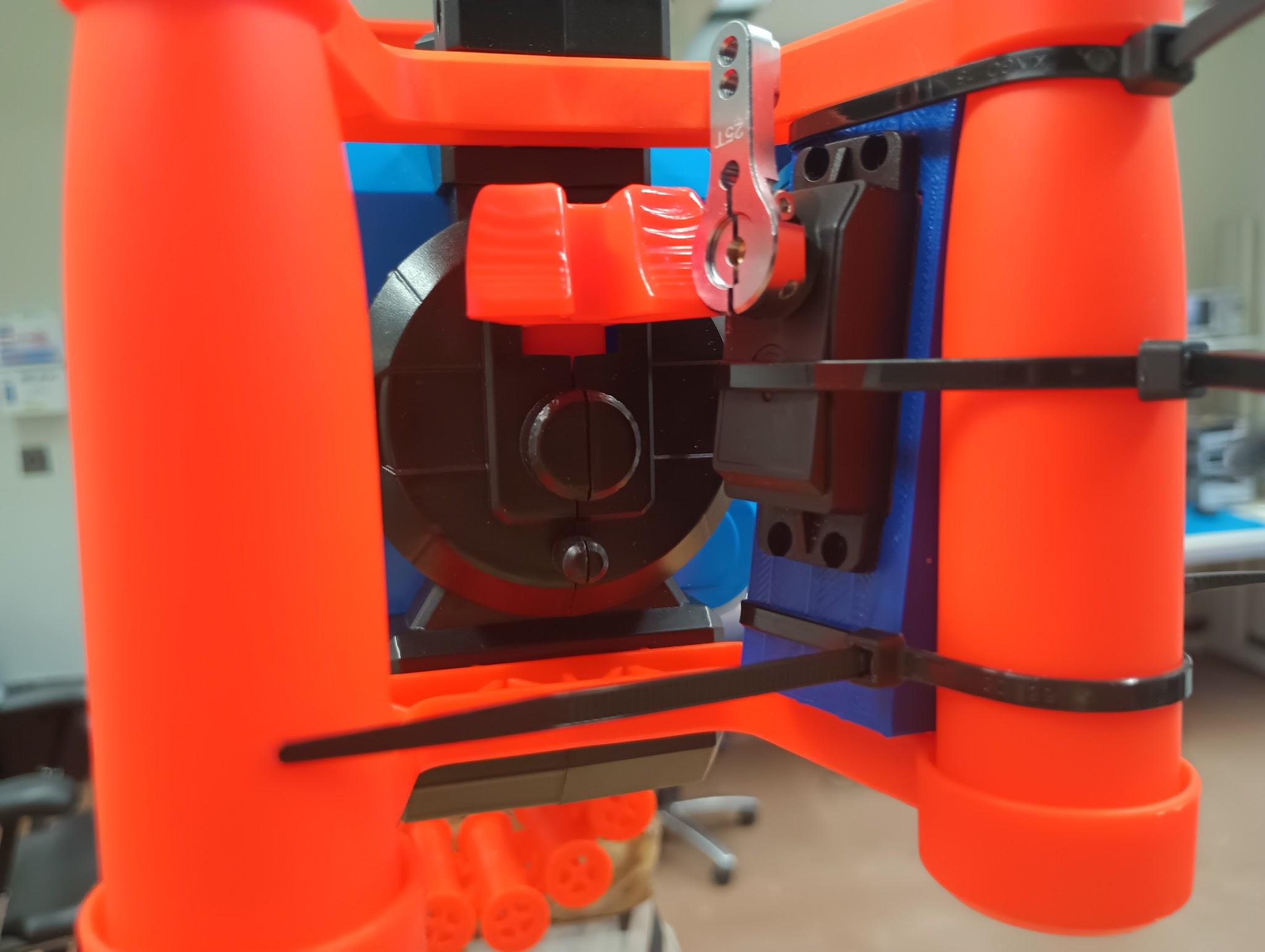


Figure 4. Servo motor mounted behind trigger

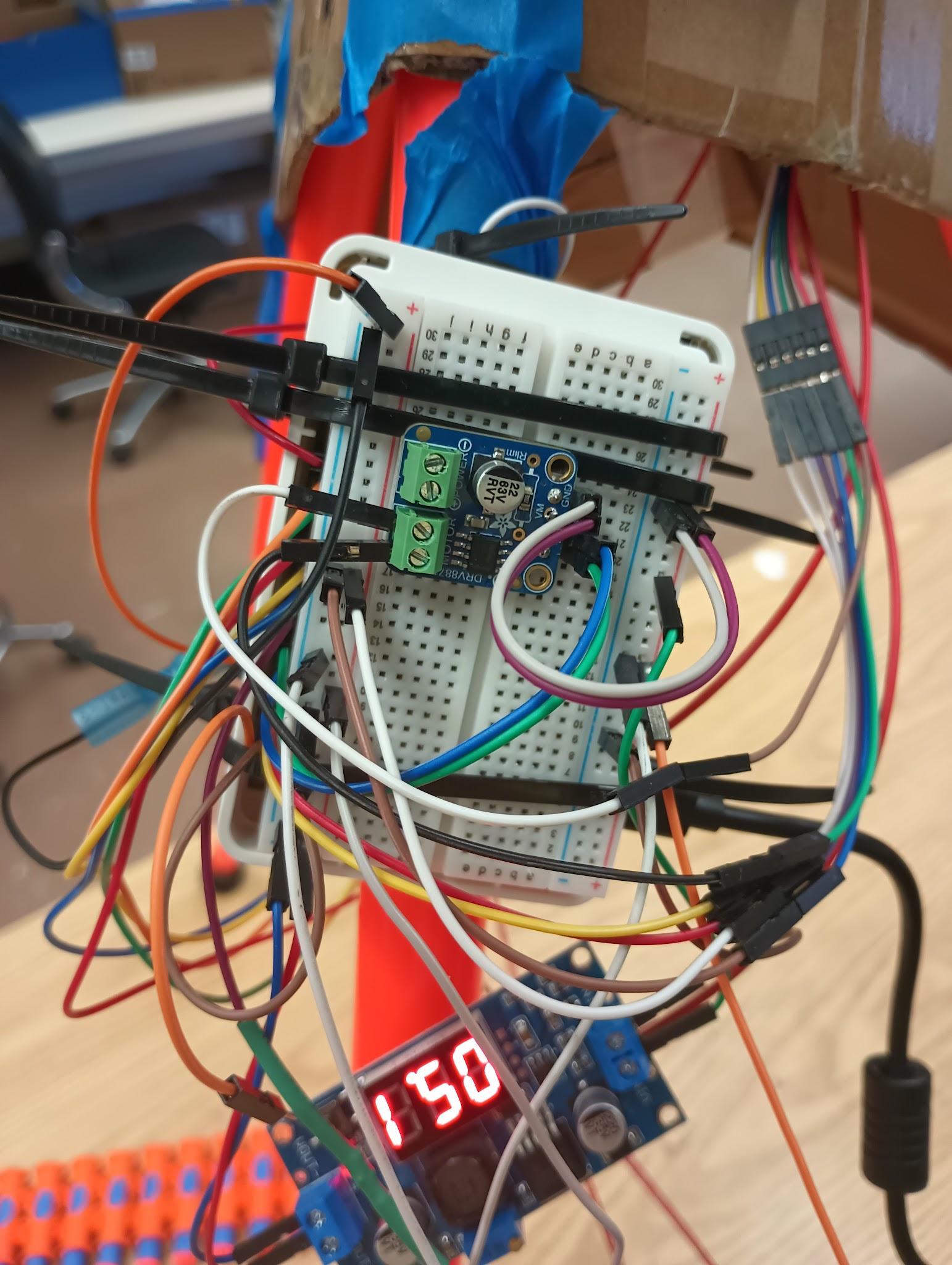


Figure 5. Electrical Components attached to tripod



Figure 6. Ring of PIR sensors and makeshift shroud



Figure 7. Final turret product

1. The code is indexed starting at 0, but for simplicity this report will index everything starting at 1 [↑](#footnote-ref-0)