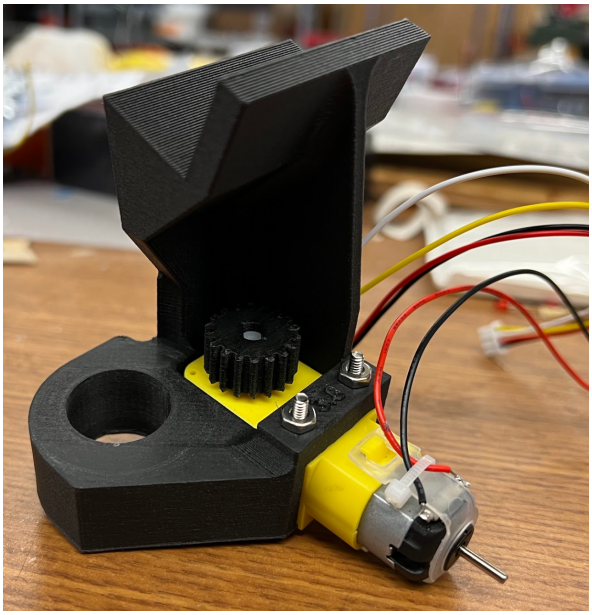
Blindr: IoT Smart Blinds

Sam Roquitte & Peter Chu

Hardware

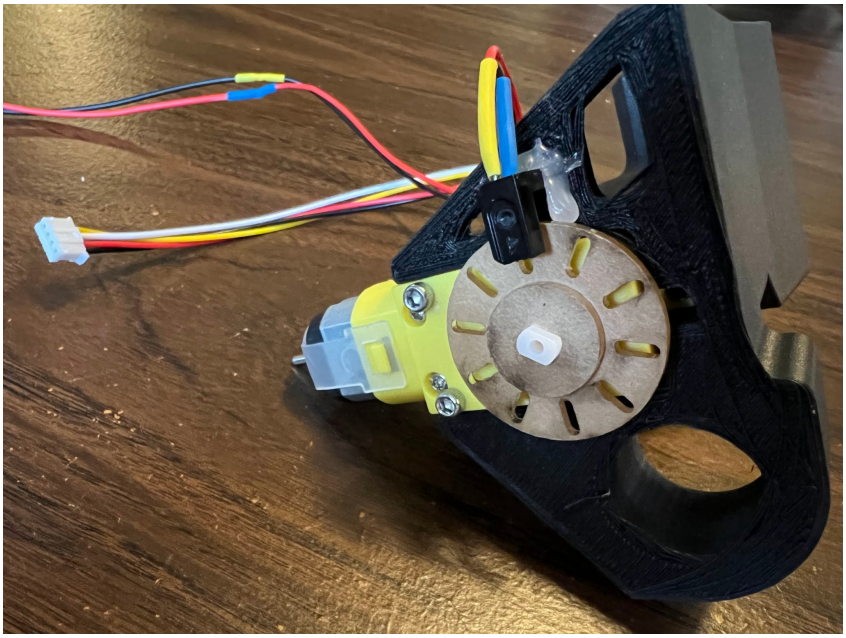
Our project is broken up into three main hardware components: the blind rotator assembly, control board, and power system. We have designed our project in three parts to allow for a modular device that can be easily customized for what the end user wants. For example, if you live in a place that gets very little sunlight (like Sam) you can choose to use a standard power supply while still using the rotator assembly and control board. Another thing that we discovered while prototyping which will be discussed in depth later is that most blinds don't have a standard turning wand. This means that the rotator assembly would need to be specific for your set of blinds and may need to be tweaked by a few degrees to fit right. By having this modularity, we can allow for these changes without needing to redo the entire project. In order to support this, we have standardized the connectors on each of the three components so that they are plug and play.

Blind Rotator



Blind rotator on table

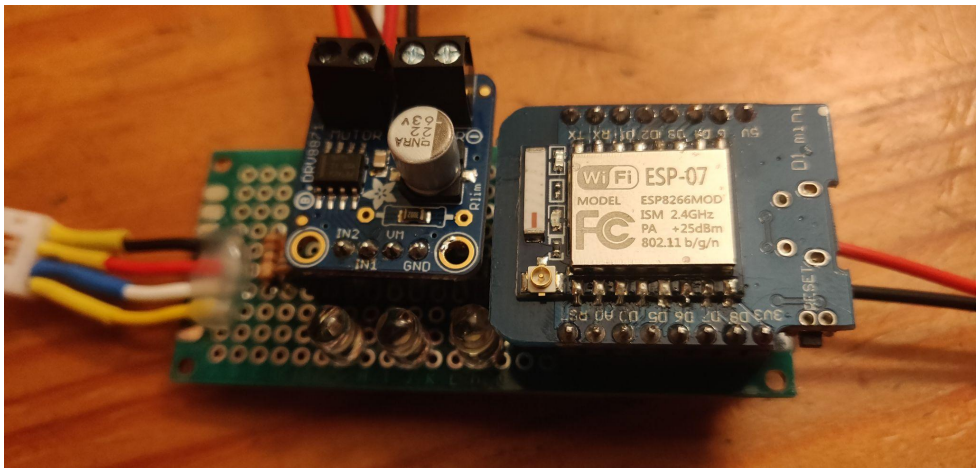
The first of the three hardware components is the blind rotator. This piece consists of a 3D printed housing that the motor can easily mount to (with two screws) as well as a few gears to drive the wand of the blinds. On the bottom side of the motor, we have mounted an optical encoder which is used to determine when the blinds have been fully opened/closed. Since we know our gear ratio is 1:2 and we have 10 slots in the encoder wheel, we need to count 10 * 2 = 20 slots for one rotation of the output shaft. In testing, we determined that 3 rotations were needed to move the blinds from fully up to fully down which means we need 1.5 to make them open (parallel to ground). In encoder ticks, this would be 20 * 1.5 = 30.



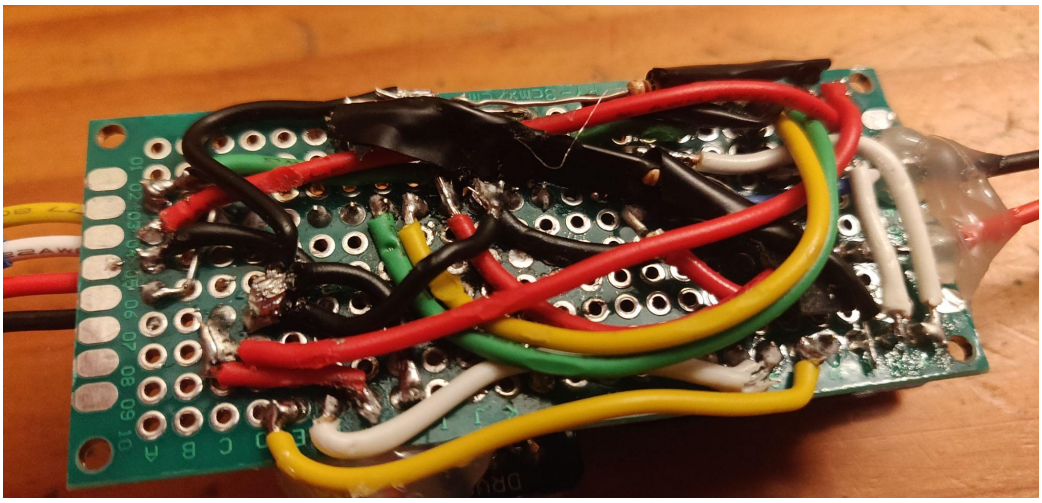
Encoder on bottom of motor

Both the motor and encoder (LED/light sensor) have easy to use connectors so that the blind rotator can be separated from the control board. In the future, alternate blind rotators could be offered to fit different blind models without the need to recreate the rest of the project.

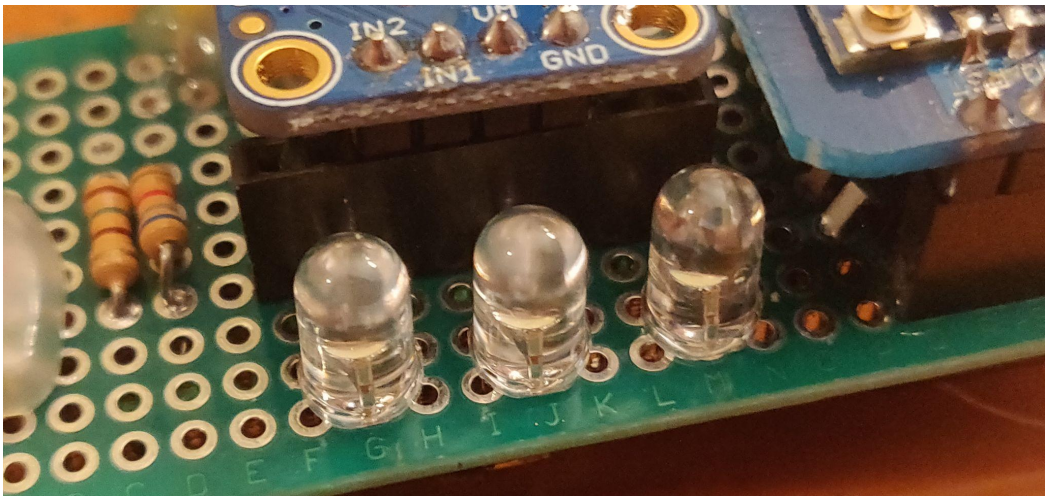
Control Board



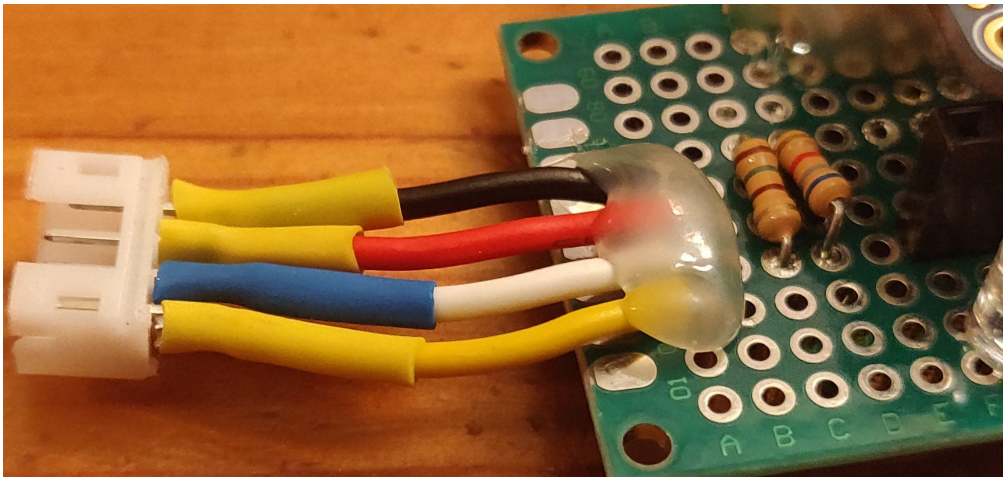
Top view of the control board



Bottom view of the control board



Close up of the LEDs on control board



Glue reinforcement for the connectors on control board

The next component in our project is the control board which includes the non power related electronics such as the microcontroller and motor controller. We chose to use the D1 mini which is an ESP8266 based board since it has WiFi built in and has very efficient power modes that can be used. For driving the motor, we originally tried to use the basic H-bridge from the skill demo but determined that 5V was not enough for the motor. After determining that, we switched to a DRV8871 which can supply 5.6V - 45V and has a 3.6A peak current.

Additionally, we added 3 LEDs onto the board to indicate the current state of the blinds and help with troubleshooting any issues. Finally, there is a provided connector for the i2c pins to allow additional sensors in the future. As an example, we have connected a BMP280 (pressure/temperature) sensor using the standard i2c connector.

The IO and pinout are based on a D1 mini with an ESP8266:

- A0 = Light Sensor (encoder)
- D1 = i2c SCL
- D2 = i2c SDA
- D4 = Top LED
- D5 = Bottom LED
- D6 = Motor
- D7 = Motor
- D8 = Middle LED

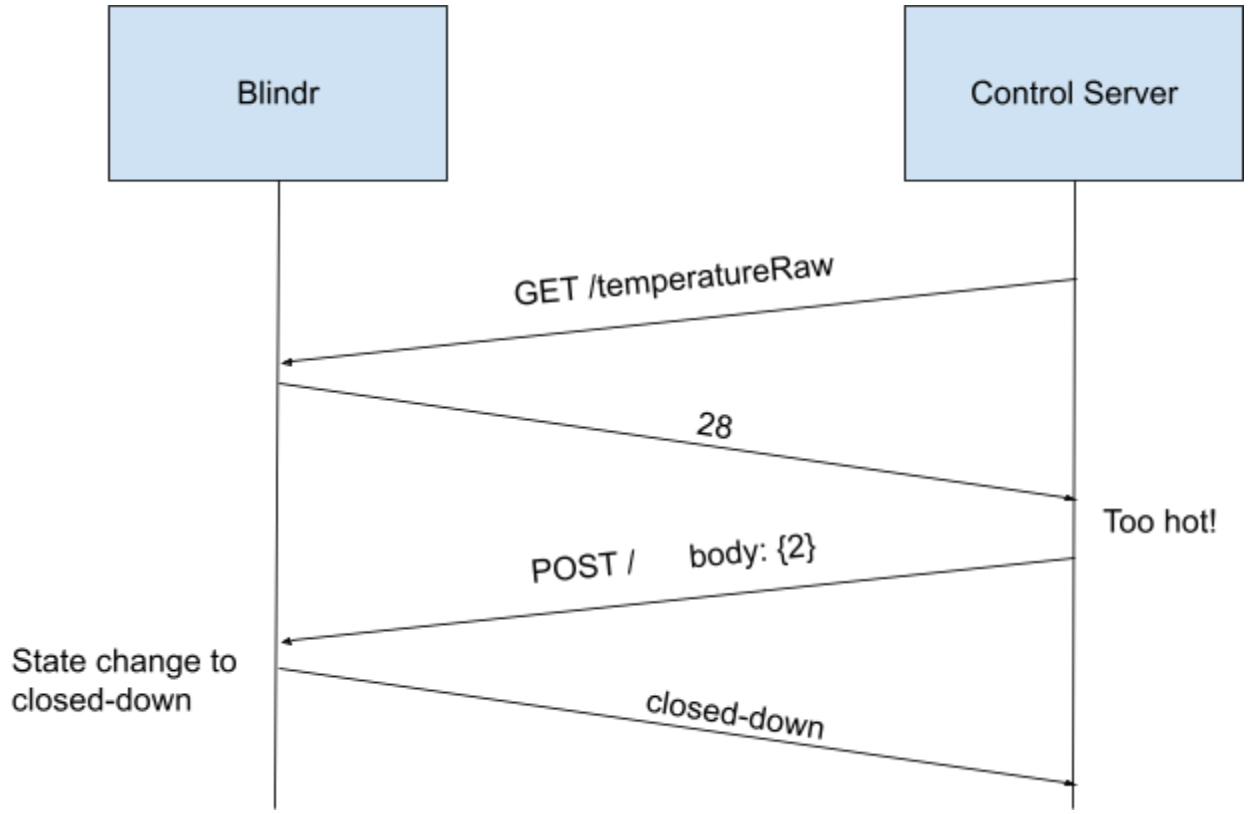
Power System

The power system includes everything used to power our project. Originally, we had planned to use a 1s lipo along with a solar panel but after running into some issues with 5V, we had to ditch this for a more traditional power supply. In the future, we could get a different solar panel & control board that allows for a 2s or even 3s lipo. This should give us the proper power to drive our motor. Due to the modular nature of our project, we can simply replace the power system with this new one and then use the connector between it and the control board. For now, due to time constraints, we will be using a 9V, 1A power supply since that is what we have available.

Code Overview

The code is divided into two parts with basic state & motor control done locally on the ESP and commands/other logic coming from the control server. This allows the code on the device to remain small and simple. It also allows for modularity with different i2c devices. In the example below, the control server is deciding the state of the blinds based on the temperature in the room. First it asks the device what the temperature is by hitting the “/temperatureRaw” route. The device responds with the temperature in C. Then the control server decides that this is too

hot and tells Blindr to close the blinds to stop letting in sunlight by POSTing to the root route with a body of “2” (closed-down). You can see that this system paradigm could work for multiple different types of sensors and control modes. Additionally, it allows for further expansion should multiple Blindrs be installed in one room. The control server can run all of the Blindrs to control the whole room or even a whole house!

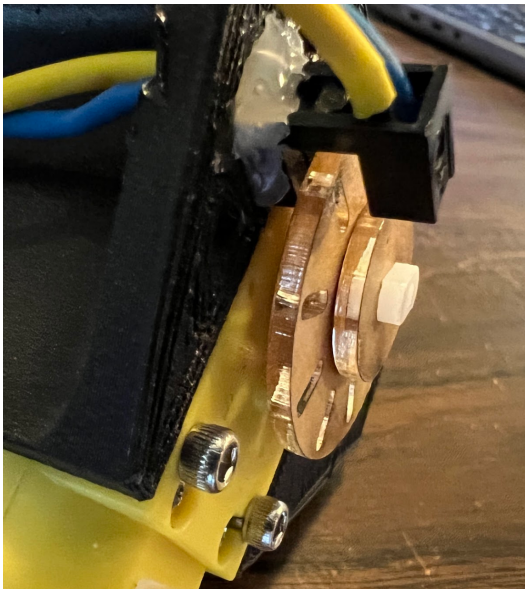


Currently, we are simulating the control server by sending raw REST requests from a computer. Since Blindr uses standard REST endpoints, the control server could be written in any language.

Libraries Used

- ESP8266WebServer
  - Host a REST server on ESP8266
- Wire
  - i2c support
- Adafruit_BMP280
  - BMP280 support (i2c temp/pressure sensor)

Parts We Designed

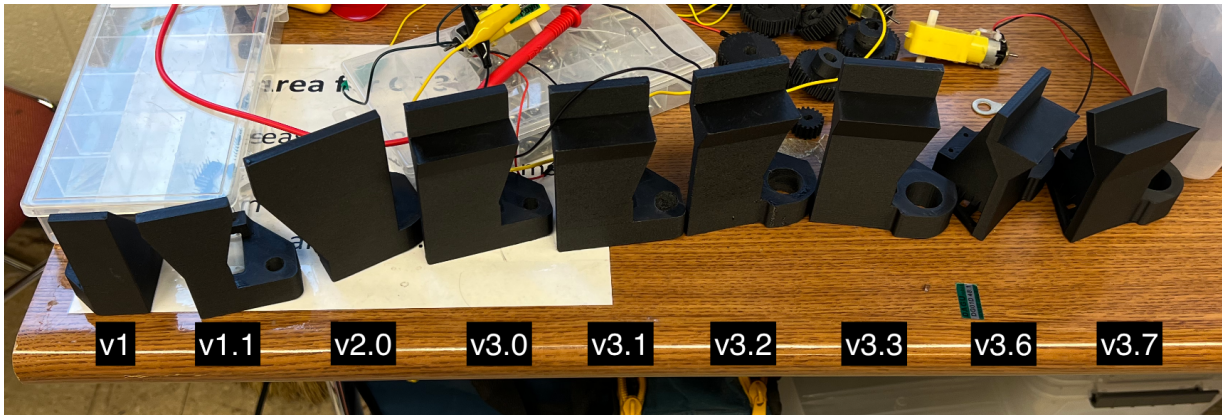
1. Blind rotator (3D printed)
    a. This piece was designed in Solidworks and went through *many* iterations
    b. It includes a cutout for the encoder sensor so that it fits tightly and is held at the correct distance/height for the encoder wheel
    c. There are also mounting holes for the motor in the design so that it can be easily attached
2. Gears for blind rotator
    a. One gear for the motor side and another for the blind wand side
    b. The motor side did not need to be modified past two revisions
    c. The blind wand side went through 8 revisions to get to the final version
        i. Originally, we were trying to turn the wand so that it wouldn't need to be removed but that proved to be quite difficult as discussed later
        ii. The final version binds to the loop and replaces the wand
3. Encoder wheel
    a. The encoder wheel & spacers were also designed in Solidworks
    b. We decided to go with 10 cutouts since we don't need super fine granularity and only need to know when 1.5 rotations have been completed (30 ticks)
    c. We needed to use a single spacer to prevent the wheel from hitting a bump in the motor housing and have it be at the right height for the optical sensor
    d. We choose to laser cut this piece out of acrylic since it would be much quicker than other methods and would be very accurate
    e. 
    Optical encoder on bottom of motor
4. Control board
    a. Includes headers for all replaceable boards (D1 mini, motor driver) in the event that one breaks (which happened to the first D1 mini)

Skills We Learned

We didn't have time to go into CAD during any of the skill demos so that was something that we had to learn outside of the course. Solidworks is a very complicated program with many different options and settings. It took quite a bit of fiddling around to get to the proper design. Knowing what we know now, we would have gone about designing it in a very different manner. For example, spending time to have accurate measurements of the blinds would have made it much easier to design. The measurements we had were apparently off by a few degrees which meant that when we tried to fit the actual 3D printed parts, they didn't fit as we thought.

With our project, we decided to use an ESP-8266 based board which was different from any of the microcontrollers that we worked with in the skill demos. It required us to learn quite a bit about WiFi since we had to get our device to talk with a server in order to know when to open/close the blinds. Luckily we had worked with REST servers in the past so that half wasn't particularly difficult but getting the ESP to make the requests was something new. You need to manually specify much more than in other languages (header, REST method, body, etc.) to send a simple request. Then when you get a response, you need to parse everything from it as a plain text string.

Iterative Process

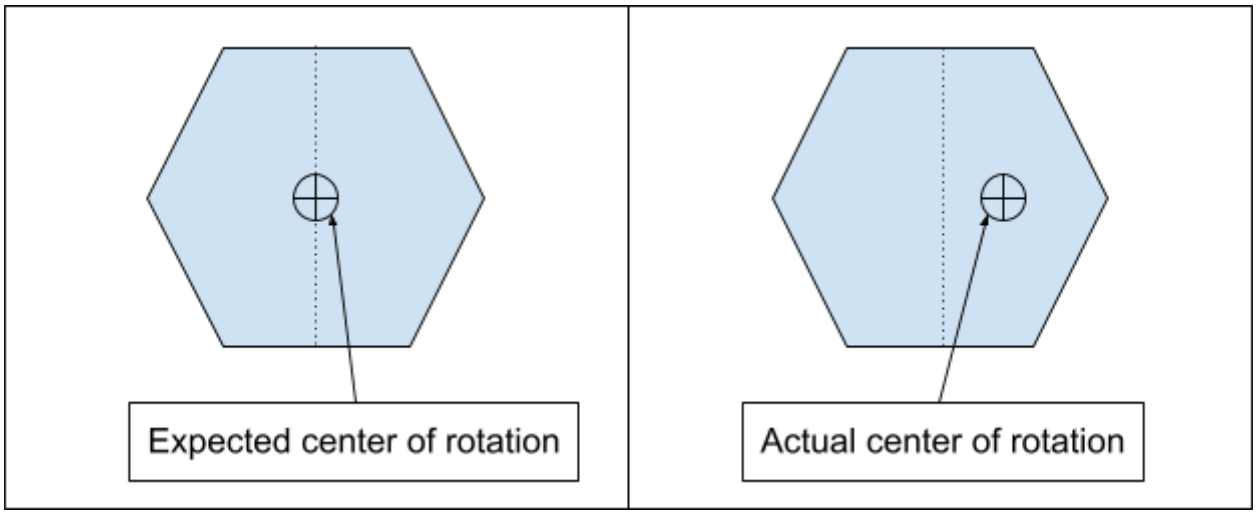


Some of the blind rotator designs we designed & tested

As you can see above, we went through *many* different designs before ending on what we currently have. The leftmost designs were earlier and ones on the right were later. Originally, we planned to leave the blind wand attached and then simply use a gear with a hexagonal hole to turn it. In the first design, we had the hole on the right side and were using a 1:1 gear ratio. When we tested this design, we discovered that the motor did not have enough torque, the hole should be moved to the left side to fit better on the blinds, and that the top attachment point was not wide enough to hold onto the top of the blinds. In the next iteration (v1.1), we corrected all of these problems and went to a 1:2 gear ratio. We found that the wand was a bit further back so

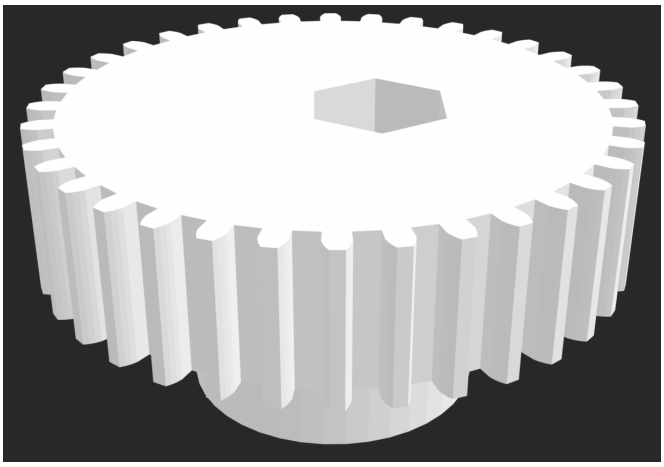
we went to correct this with the next version. v2.0 was also not in the right spot so we needed to try again in the next version. Additionally, we thought that we could make a better connection with the mount point by including a L shaped piece at the top to connect to two sides of the blinds instead of just one. In v3.0 and 3.0, we tried to address these issues.

With 3.1, we realized that the blind wand doesn't turn the way we thought it did. The center of rotation was not actually in the center of the wand, it was a bit outside of that. See the diagram below for further explanation.



Blind wand center of rotation

To solve this new issue, we attempted to design some new gears that had an offset hexagonal hole (see below). Along with this, we put a larger cutout on the blind rotator so that it could slot in there and allow for the outside of the gear to have a proper center of rotation (so that it can mesh with the other gear). We tried 4 different offsets of the hexagonal hole (1-4mm offset from center) but none of them worked very well so we scrapped this idea in the next iteration.



Gear with offset hexagonal hole

This brings us to v3.6 which used a new type of gear that instead of connecting to the wand, connects directly to the metal loop on the blinds. While testing this version, we started to

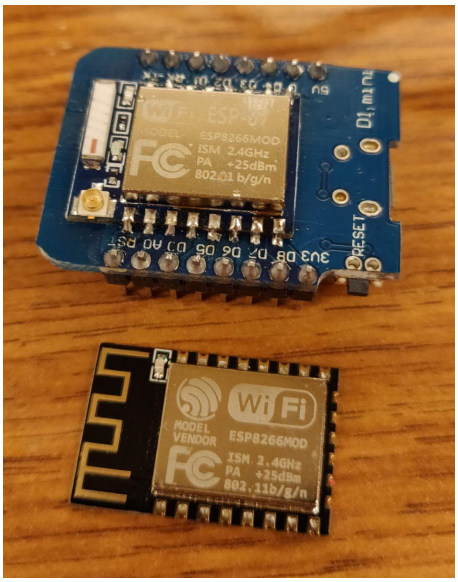
see signs of success. The blinds were starting to turn but we noticed heavy resistance on the motor and it would get stuck at certain points in the rotation. With this iteration, we tried holding it at different angles and noticed that it turned much better when held back at about a 30deg angle.

Gear connecting to metal loop on blinds

In the next revision, we angled the mounting piece back by 30 degrees and noticed a large improvement. It wasn't at the perfect angle though (see image above) so we had to do another iteration to correct that. In the final revision (v3.8), it was at the proper angle and the blinds were able to turn!

Once we had the blind rotator finalized, we noticed that the motor was much more powerful & reliable when supplied with higher voltages (around 10V seemed to be best). Unfortunately, we realized this after soldering all of the components (H-bridge included). We found a new motor driver which had a much higher power allowance and replaced the standard H-bridge with that.

Problems Encountered



Picture of replaced ESP device on breakout board

Picture of successful replacement of ESP reaction with professor

During the soldering of the control board, there was an exposed wire that connected to power voltage. That wire connected to a digital pin to the ESP and killed the whole ESP. Morale was low especially at 1am in the classroom with a dead ESP. Then the professor steps into the classroom at 1AM. After an awkward long stare between the professor and Peter, the professor and Peter troubleshooted and performed tests such as powering the 3.3v line of the ESP to make sure it was the ESP was actually dead. Because the breakout board was drawing half an amp, the professor and Peter determined that indeed the ESP died. Then Peter went into the professor's car to the TSRB lab in Tech Square and used their hot air reflow station to replace the ESP on the breakout board.