Wireless Controlled Prosthetic Arm

Tejus Kurdukar: tkurdu2, Taylor Xia: tyxia2

Github link: https://github.com/TheRealTaylorXia/Ece-110-Honors-Lab-Project

Purpose

Our goal was to construct a prosthetic arm capable of responding and interacting with the user. The greatest benefit of our design is the added aspect of wireless communication between the user and the arm, removing the need to be attached to the apparatus while still enabling accurate and precise mimicking of the user's hand.

Design Details [1]

Arduino Unos

- Used to control servos and power transceiver on Arm-side
- Reads and maps values from flex sensor to motor angles and sends data on Hand-side

Flex Sensors

- Analog sensor that has varying resistance depending on how much it's bent
- Resistance values are read, mapped to angles that are writable to servos, and transmitted

NRF24L01+ Transceivers

- Transmit and receive radio signals from one Arduino to the other
- Radio signals contain mapped flex sensor values

MG996R 55g Micro Servos

- Total of 5 used to control each finger on the hand
- Attached to servo bed inside of arm

3D printed Arm Assembly

All parts were designed and obtained from Gael Langevin's *InMoov* project, an open source 3D-printable robot design with provided STL files. The wiring of the fingers involved two strings: a front string made of upholstery threading is used to pull the finger down, and an elastic material (rubber or nylon) to string is placed in the back of each finger to pull the finger back up. The two strings are then fed into a pulley system of five servo motors, designed so that rotating the pulley from its 0 position will pull the finger down and moving back to the 0 position will allow the finger to move back upwards [2]. Each servo is finely tuned in order to responsively move its corresponding finger - the tensions of the elastic restoring strings are adjusted to quick response times and the pulleys themselves are slightly offset to move the fingers down more immediately.

Transmitter-Side Code and Circuitry [3]

Each flex sensor is connected to the power and ground rail on the breadboard, with the power rail being given 5V from the Arduino. Attached in series with each flex sensor is a 10kΩ resistor to create a voltage divider - considering that a flex sensor is simply a variable resistor and that

the Arduino can only read voltage values, a voltage divider is necessary to be able to read the voltage drop across the flex sensor. Then, after each flex sensor is attached to analog pins of the Arduino, the Arduino can use this value to calculate the sensor's resistance using Ohm's Law. Analog pins output a number, somewhere between the range of 500-200 but each sensor is unique in the code. These values are mapped using a function, available from the default Arduino library, to a range between 0 and 180, the minimum and maximum angular range of the servos. Once that's done, the mapped values are stored in an int array and sent. [4]

Receiver-Side Code and Circuitry [5]

To power the motors, we connected them in parallel with an independent voltage source, using two 3.7 V batteries. The motors are then connected to five pins of the Arduino, which will be able to send electrical signals to the motors to change their angles. The NRF transceiver is connected to the Arduino to receive values from the transmitter side that are used to control the angles of each servo

To establish the connection between two NRF transceivers, it is necessary in code to set them both to communicate on the same address and "pipe." At a high level, this process is analogous to connecting the two transceivers on the same network channel, so that they know to communicate with each other and no one else. Once they are synchronized with the same data rate and clock, the transmitter is initialized to begin writing information on the pipe and the receiver is set to read data. [6]

Results

To first test that our NRF modules were connected properly, we ran a program found in the NRF library which included multiple instances of example code. The program would run through and simply output a series of hexadecimals in the serial monitor, indicating whether the Arduino was properly connected and speaking to the NRF. We then used a different example program to send a simple string message back and forth between the two Arduinos.

Next, we ran some very short and simple servo test code to test each finger and it's complete range of motion. The program simply started the servo off in the 0 degrees position and had a for loop that would increment the angle by 10 until reaching 180 then decrement it back to 0.

Finally, we assembled everything together to have each of the fingers move independently based on the transmitted data values from the flex sensors and Hand-side Arduino. Unfortunately, we weren't able to complete the rock-paper-scissors portion of our goal, however that was only due to a lack of time; the fingers of the hand are fully capable of reading and producing the paper, scissor, and rock motion. In other words, the hand is physically capable of playing, only the code aspect is missing. Overall, then we concluded that we had succeeded in creating our synthetic hand and having it respond and interact with the user.

Problems and Challenges

NRF24L01+ Transceivers, Arduino UNOs

The process of getting the wireless communication system to function properly was the most time consuming and frustrating part of this project by far. Using the connection test tutorial code we had obtained online, we greatly struggled to get the transceivers to send even simple string messages to each other. There were several things we did to troubleshoot this problem:

1. First, we read online that NRF transceivers occasionally have issues with receiving the correct consistent amount of power from Arduinos when outputting 3.3V. A common solution to this is to attach ~10uF capacitors to the Vcc and GND terminals of each transceiver in order to smooth out voltage and prevent dips and spikes. We attempted this, then attempted 100uF capacitors, however neither solution fixed the problem.
2. Second, we began to wonder if the problem was due to software, and we tried running multiple different test codes, running our own, and researching the library more in depth. One user online suggested inserting the line: radio.setAutoAck(false);. This had some effect on our problem: finally, the transmitter was claiming to be properly functioning and was stating that the receiving end of our system was acknowledging the proper signals. Frustratingly however, the receiving end of the system was unable to output the supposedly transmitted data and was asserting that the transmitter was unavailable.
3. We attempted to replace the transceivers with new chips, which did not fix the solution and indicated that the transceivers were not faulty.
4. We thought at this point to test the communication between NRFs and their Arduinos despite many retries at rewiring. We found test code online as described in our results section that is used to verify that the Arduino is interacting properly with its NRF. After running the code, we found that the Uno was communicating properly with the chip, but the Nano was not outputting anything to the Serial monitor.
5. Because of our suspicion of the Nano not outputting anything about the module, and after reading a few hundred more forum posts, we suspected that the problem was simply a matter of a faulty Arduino. We ended up rushing to procure another Arduino Uno, which finally fixed the problem.

The issue of troubleshooting the wireless communication was a process that took several weeks of our time. Because of this, we unfortunately did not have much time to elaborate as much on the project as we would have liked to.

Flex Sensors and MG996R 55g Micro Servos

The Honors Lab inventory only had 2 flex sensors when we had searched around at the start of the semester. We ordered 3 more, but the order form was unfortunately either lost or didn't go

through. In the end, we were forced to order the 3 other sensors once returning home from Spring break, causing our construction to be delayed by about two weeks. The servos gave us a similar problem; we had gotten the wrong king of servos from the lab inventory (9g instead of 55g). We also had to order those once we got home which contributed to the delay.

3D printed Arm

Though InMoov had provided the STL models for our arm to be 3d printed, the instructions he provided on his website for construction were vague at some parts and overly complex in others. Thus, most of the arms construction was improvised by us during assembly. As one may expect from an improvisatory wiring and assembly process, we encountered a multitude of small problems during construction. The tension required for each string was found entirely through trial and error, and at times it would take several hours to adjust one string correctly. There were many 3D-printed parts that were either slightly warped, had holes that were too small, or didn't fit well together, which required sanding and drilling to fix. There were also several string breakages over the course of our assembly which required repeating things we had already done, making this a very arduous process at times. All of these issues, coupled with some more minor problems such as screws fitting or gluing errors, made assembly a very time consuming process.

Future Plans

There were a good number of aspects to this project that we could not complete due mostly to time constraints. In the future, we're hoping to perhaps program in a functioning version of the rock-paper-scissors game onto the Arm-side Arduino. With that, we'd also need to actually create the glove controller itself by finding an efficient yet nice looking way to attach all of our circuitry to a reasonably sized glove. We may also look into replacing some of the rubber tubing we'd used as our elastic string with nylon, as the tensile strength of that material seems to be much greater than that of the rubber we've currently using.

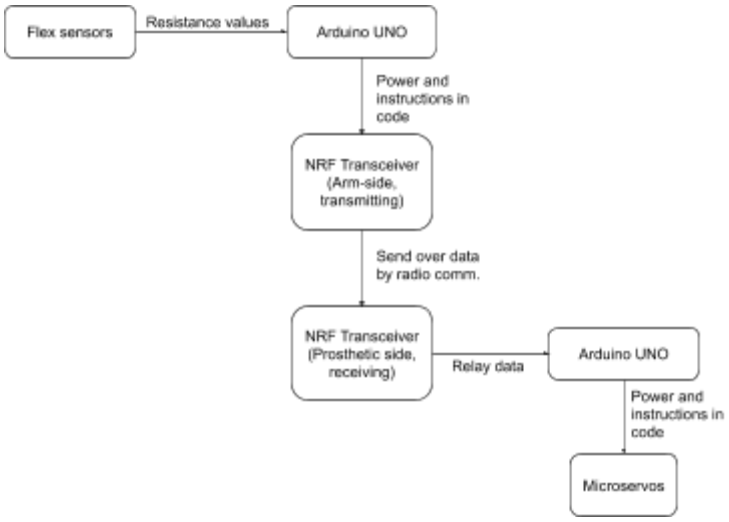
References

[1]M. Kilic, "How to Make Wireless / Gesture Control Robotic Hand", *Hackster.io*, 2018. [Online]. Available: https://www.hackster.io/mertarduino/how-to-make-wireless-gesture-control-robotic-hand-cc7d07. [Accessed: 15- Feb- 2020].

[2]Gael Langevin, "Hand and Forearm", *inmoov.fr*, 2014. [Online]. Available: http://inmoov.fr/hand-and-forarm/ [Accessed: 11- April- 2020].

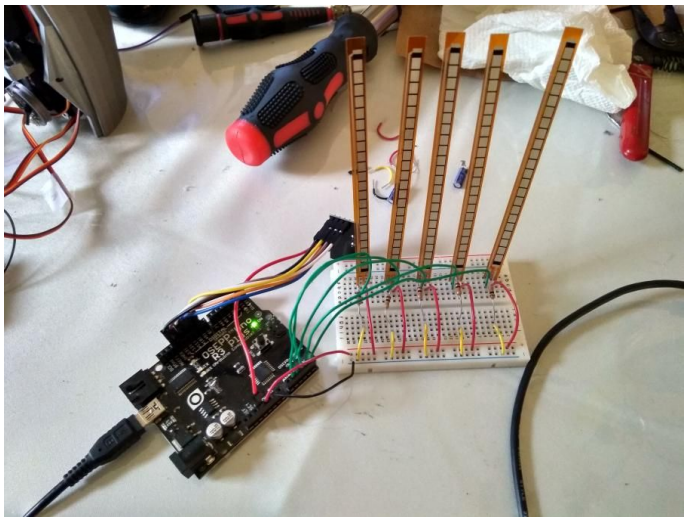
Appendix

[1]

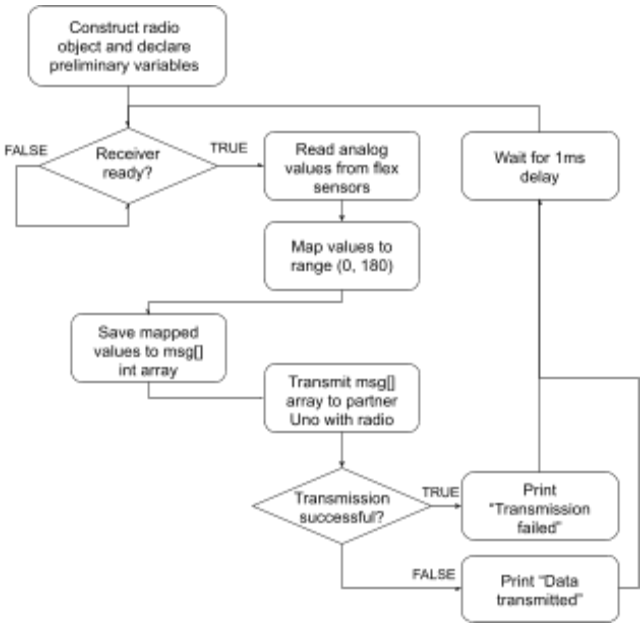


[2]

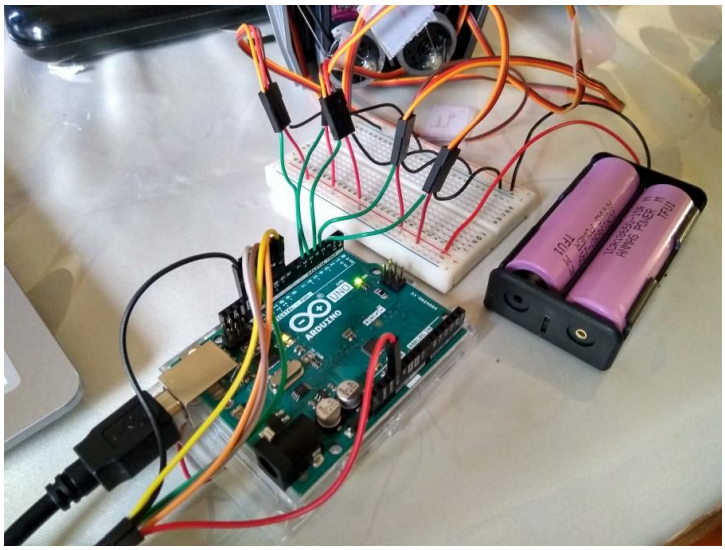
[3]



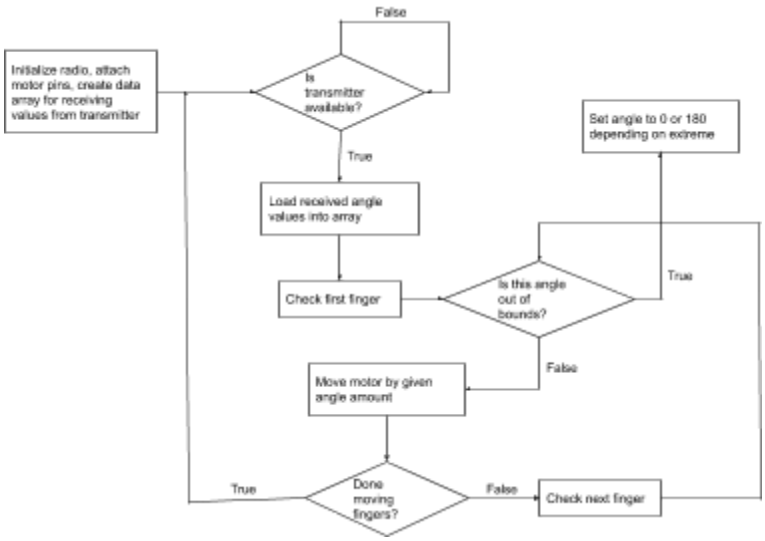
[4]


Construct radio object and declare preliminary variables
Receiver ready?
FALSE
TRUE
Read analog values from flex sensors
Wait for 1ms delay
Map values to range (0, 180)
Save mapped values to msg[] int array
Transmit msg[] array to partner Uno with radio
Transmission successful?
TRUE
Print "Transmission failed"
FALSE
Print "Data transmitted"

[5]



[6]


Initialize radio, attach motor pins, create data array for receiving values from transmitter
Is transmitter available?
False
True
Load received angle values into array
Check first finger
Is this angle out of bounds?
True
Set angle to 0 or 180 depending on extreme
False
Move motor by given angle amount
Done moving fingers?
True
False
Check next finger