Free Motion Robotic Arm

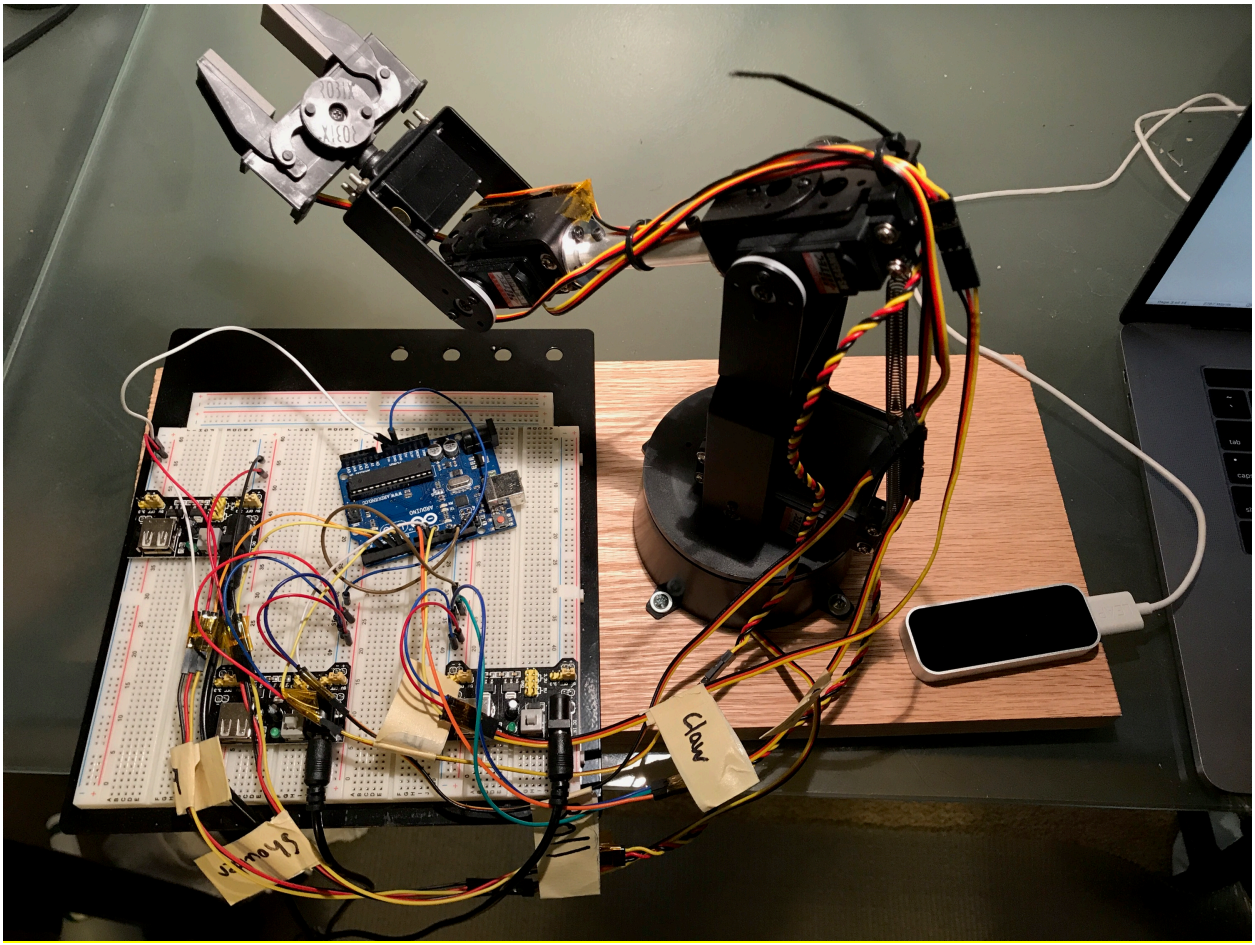
Vincent Tran

Table of Contents

Introduction

The whole idea of this project is to get a robotic arm to closely mimic the movement of your hand in real-time. I wanted to stray away from the typical use of joysticks to control the robotic arm. With the use of Leap Motion, the palm of your hand and fingertips are closely measured in a 3-D plane. The reflected data is sent to the host computer from where I run JavaScript code to manipulate the data to implement the functionality for the robotic arm. The data then gets sent to an Arduino Uno via serial communication to drive the servos of the robotic arm.

There are 5 degrees of freedom on the robotic arm, up/down, left/right, forward/back, pitch, and roll. The controls are straight forward. Treading along the x-axis moves left and right, and the same idea for the y and z axis for up/down and forward/back, respectively. Tilting your hand along the pitch raises and lowers the wrist of the robot, and tilting along the roll twists the wrist. The claw functionality also works by closing and opening a fist.



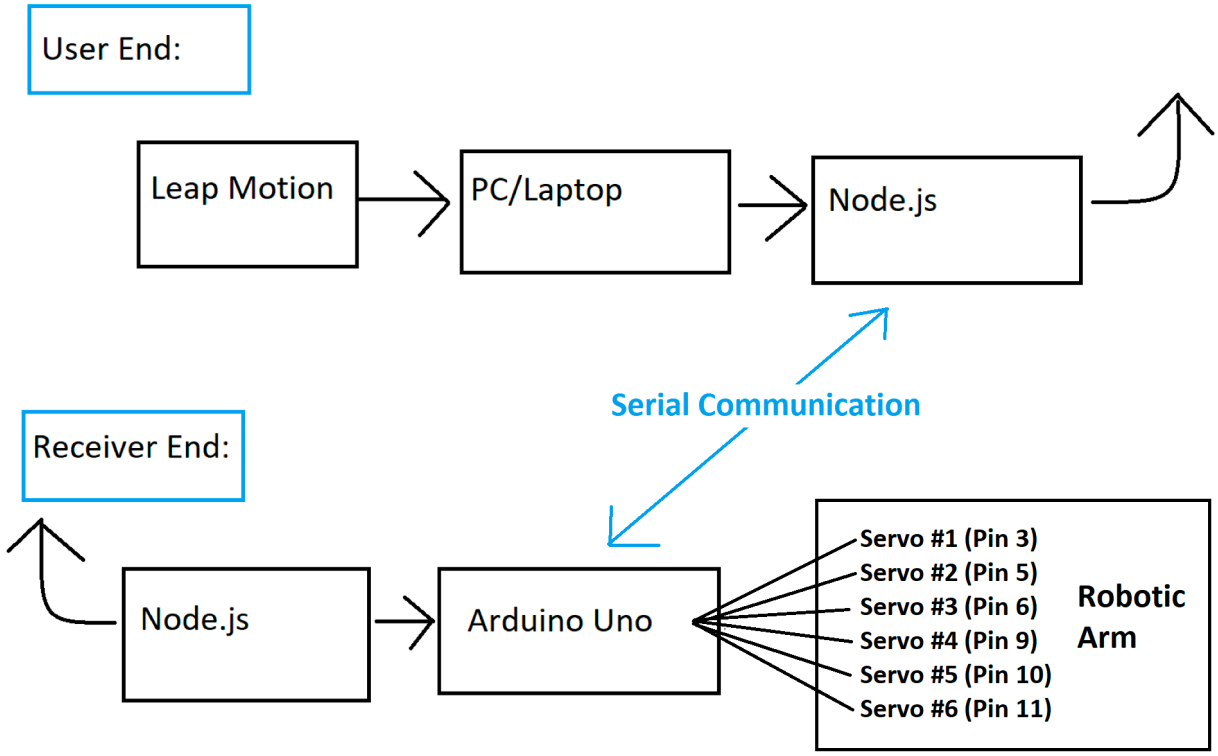
Hardware

Parts List

The hardware that was **used** in this project is listed below. The equipment that was not taught in this course has been bolded. *Include part numbers when available*.

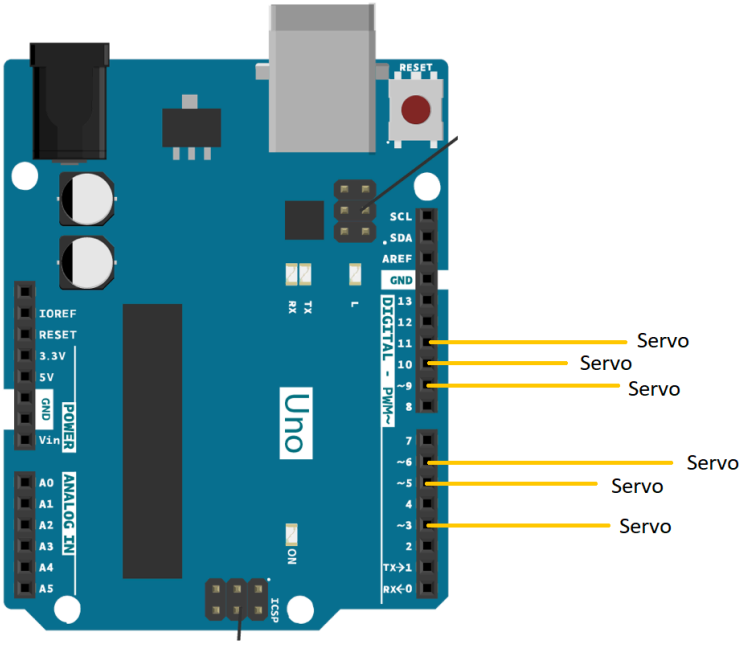
Part	Part #	Quantity	Price (optional)
Mac as the host machine		**1**	
Arduino Uno		**1**	$5 (Ali-Express)
Robot Arm Kit	**AL5B**	**1**	$120
Leap Motion		**1**	$60
Servo	**HS-485HB**	**2**	$20~$30
Servo	**HS-645MG**	**1**	$10~$20
Servo	**HS-755HB**	**1**	$10~$20
Servo	**HS-422**	**1**	$10~$20
Servo	**HS-225MG**	**1**	$10~$20
		Total	$245~$300

Block Diagram



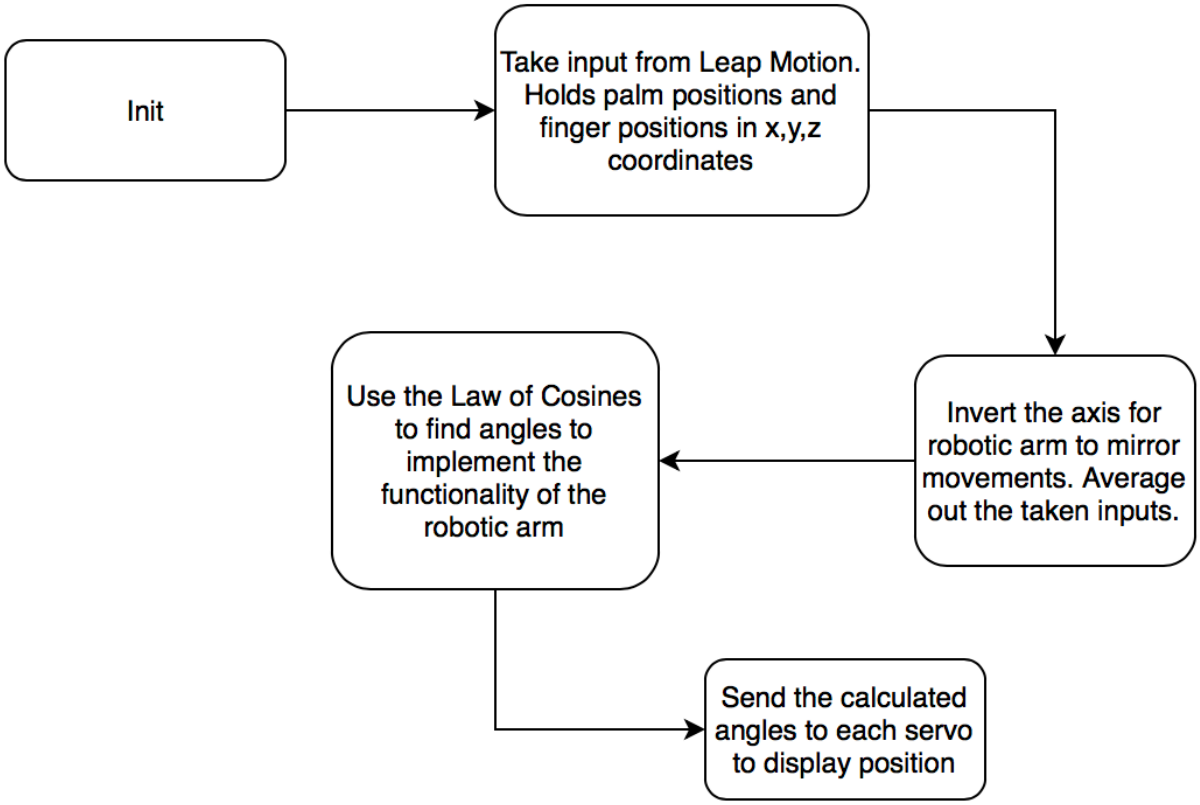
At the user end, I am interfacing the Leap Motion controller with the JavaScript program which is on a host machine that runs Node.js.
At the receiver end, I am using the Arduino for interfacing the outputs of the robotic arm. Data is transmitted serially.

Pinout (For each microcontroller/processor)



Software

The software designed for this project was implemented using the PES standard. The overall design as a task diagram is included below.



Implementation Reflection

After I received a green light to do this project, I immediately started looking at JavaScript examples. Luckily, it wasn't too difficult learning how to access objects and their properties. I didn't need to do anything too far out of my scope either, especially because I consider myself a "generalist" when it comes to making projects as an EE major. Afterwards, I took a look at the Leap Motion API to get a general idea of how I would be able to approach the project. After gathering all the tools I needed, I struggled to start. I just didn't know where to start and how to piece everything together. I spent a lot of time getting nothing to work, and I was close to proposing a new project. It didn't help that I was freaking out either, but I kept on trying. I was stuck because I thought I understood objects in JavaScript, but I really didn't because there are ways to get valid objects from another object, etc… However, with a lot of researching and test trials, I understood which objects I can access from which class from the Leap API. Everything began to work out smoothly after I truly understood how to use JavaScript.

I believe I nailed the project, given the amount of time I had to learn JavaScript syntax, and to get the functionality of the arm. The robotic arm works very efficiently and doesn't bug out if you are controlling it as intended. I am very proud of what I have accomplished and I don't think it could have come out any better, again given the amount of time I had. What I like the most about my project is that it proves that math is always right. The geometry that we learn in classes and rarely outside of it was put to good use. All the angles change in a triangle when at least of the angles change, and it is possible to get undefined angles given the lengths of the sides. My project projects math usually done in 2D onto a real 3D model, which can also be used to portray a real simulation of how the angles in triangles change. Not only that, the whole idea of having a robot copy your movements is just plain awesome. It can also grab stuff, so it is quite dynamic. I also like how my project can be deployed for more serious aspects such as bomb diffusion.

If I were to do my project again, I would probably implement cooler features. With the use of the Leap API, I would be able to read gestures such as drawing a circle, or a simple swipe. With those I can make the arm dance, or practically do anything when a gesture is sensed. The math never lies, which means when angles and lengths become smaller than others, the rate at which the new angles change may be very fast, which means the robotic arm becomes very sensitive. I would probably create new cases for the arm to never reach that position and therefore having the arm not deal with that problem. A very ambiguous task I can also do is try to create my own protocol for communicating with a microcontroller from software on a host computer, but that would be another project itself.. but in this case I just used the "johnny-five" API for the protocol along with Arduino's firmata library.

Milestone

My milestone was to bridge the leap motion and Arduino together, and to get any of its functioning down, which would require me to learn how to code in JavaScript as well. When I got checked off for my milestone, I barely fulfilled what I needed. I spent a frustrating 2-3 weeks just to take the x coordinate of my palm position and had one degree of freedom working. I would actually consider the milestone to be a little more on the ambiguous side because there was a big learning curve to it all. But to be fair, the project itself is big, so if I met the big milestone I can't complain. It was just right because on the day of the milestone check off, I had a good understanding of what I would be able to do next to get the 80-90 and 90-100.

Completed components

I was able to get my 90-100 which was having everything function correctly and smoothly. When everything is put together, the arm moves in 5 degrees of freedom which all interacts with each other. The logic of the inverse kinematics was there, and the functionality is good thanks to the law of cosines. Thanks to the "johnny-five" API and the firmata library on the Arduino, I was able to bridge the JavaScript executed code with the Arduino to drive the servos of the arms.

Incomplete components / Problems

The complete arm functions smoothly as intended. One of the problems I ran into was the origin of the leap (0,0,0) was right on top of the sensor itself. I had to offset the origin for intended use. However, as I progressed through the project, I had to keep tweaking the origin, and I had to constantly remap the x, y and z values in order to get the angle values I needed to send as a position value on each servo (0 – 180). It was very annoying because the I had to convert from radians to degrees, so it wasn't a 1:1 adjustment where it was simple as adding any value.

Youtube Links

Make sure they are publicly viewable!

- Short video: https://www.youtube.com/watch?v=zXnYuIUx5W8
- Longer video: https://www.youtube.com/watch?v=O8ReIFXLpFM

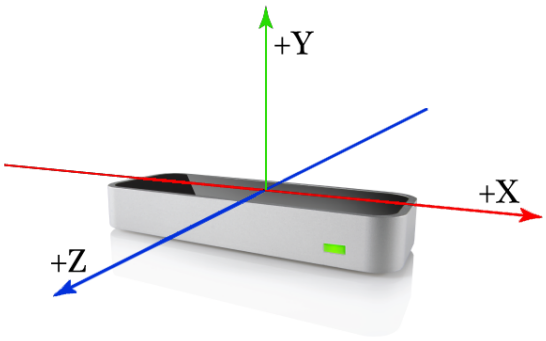
Testing

The way I tested my project was by implementing one servo function at a time. I personally tested everything myself while progressing through, until I finished. When everything was complete, I let my 5-year old cousins try it, and my relatives of all ages with different hand sizes give it a try. While observing them, I took into account the values I needed to change.
*Down below was my testing process for each feature.

Offsetting Coordinates

Throughout the whole process of implementing my project, I had to constantly change the offset of the coordinates to correct the functioning of my robot arm. To do so I would just add or subtract a value to the given coordinate. Again, these values kept changing. I also had to invert the z-axis to get the arm to mirror the input.

```
//y - 0
//offset y
frame.hands[0].palmPosition[1] -= 85//65;
console.log("y:"+frame.hands[0].palmPosition[1]);

//z - 2
//offset z
frame.hands[0].palmPosition[2] = (-1*frame.hands[0].palmPosition[2]) + 120;
console.log("z: "+frame.hands[0].palmPosition[2]);
```



Base – left/right movement

At first, I tried implementing the left/right movement by only taking input from the x-axis.

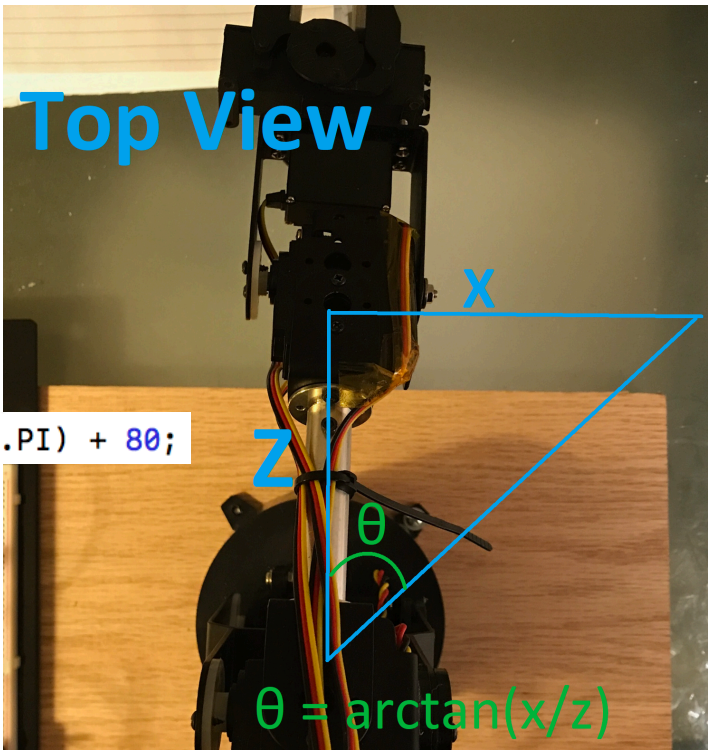
```
base_pos = xAvg * (180/Math.PI) + 80;
```

After I tested this position I realized that this wasn't right because it was too sensitive, and also if I were to reach to the right corner, my robotic arm would already be to the very right.

Therefore, I decided to work with both the x and z axis, by taking the arctangent of x and z.

```
base_pos = (Math.atan(xAvg/zAvg)) * (180/Math.PI) + 80;
```

This correctly outputs the left and right movement of the arm because it works with the angle as seen in the picture.

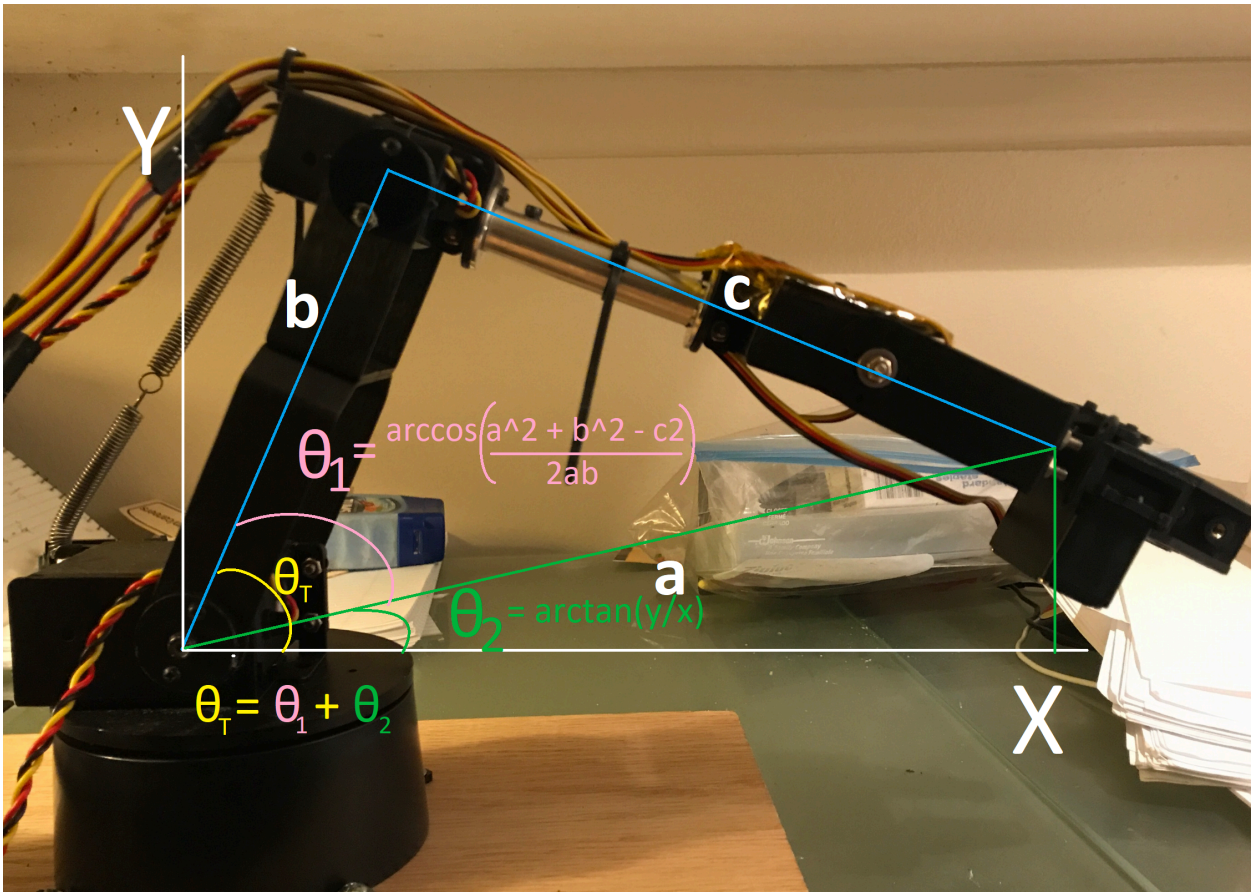


Shoulder – Forwards/Backwards movement

First of all, getting the hypotenuse is very important because it will be constantly changing, and we will be using the hypotenuse to calculate other angles as well. The hypotenuse is formed by the x and y values, and using Pythagorean theorem.

This movement consisted of adding two angles together as shown below. The math in this case was very straight forward. It functioned correctly when trying to move forwards and backwards.

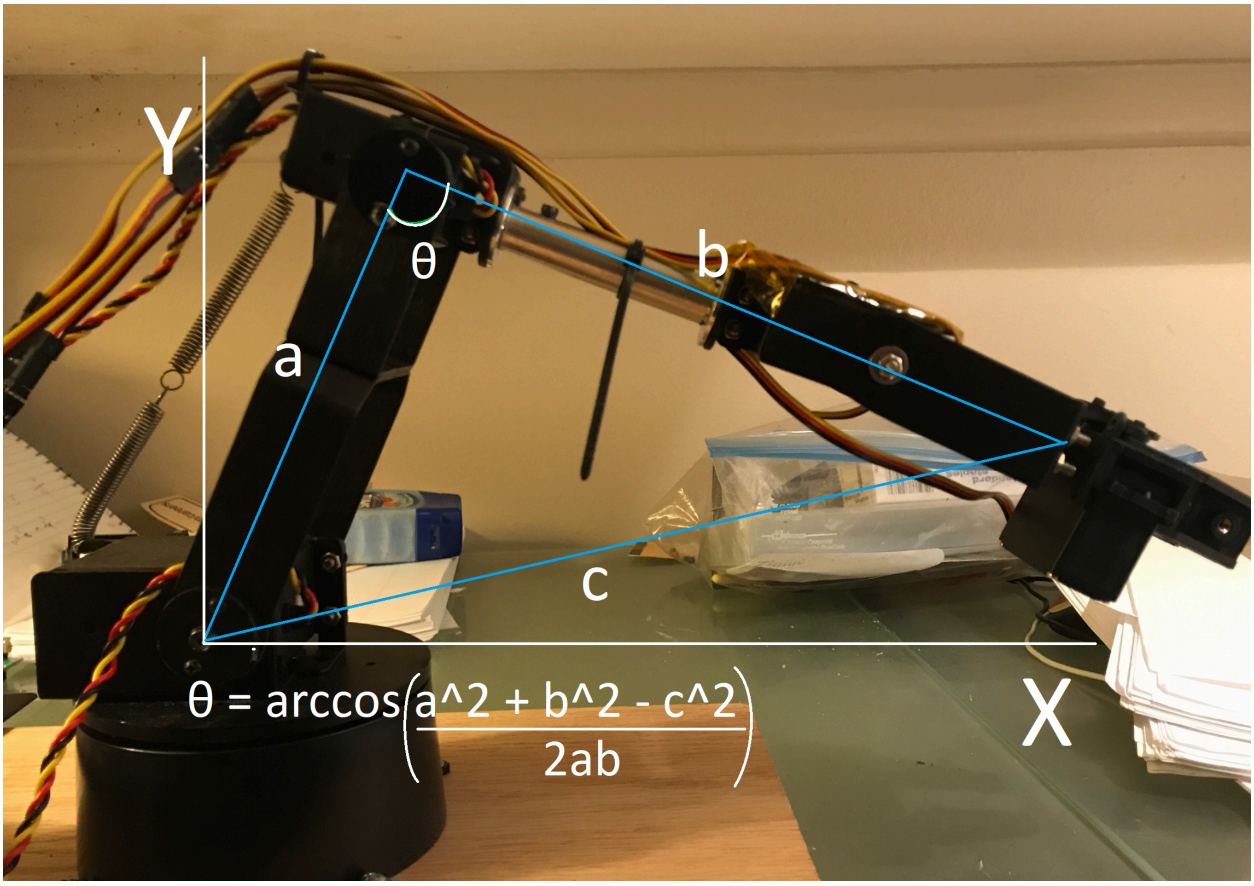
```
hypotenuse = Math.sqrt(square(y)+square(z));

//atan2 takes care of both positive and negative cases
L_base_angle = Math.atan2(y, z);
U_base_angle = Math.acos((square(hypotenuse) + square(L_arm_length) - square(U_arm_length)) / (2 * hypotenuse *
    U_arm_length));

// First angle
var shoulder_angle = (L_base_angle+ U_base_angle) * (180/Math.PI);
```



Elbow – Up/Down movement

This angle was also very straight forward. In order to calculate it, I used the law of cosines as seen in the picture below. I noticed that the smaller the servo value, the wider the angle got, therefore I started off with the position at 180 and subtracted the angle.

```
var other_angle = Math.acos((square(L_arm_length) + square(U_arm_length) - square(hypotenuse)) / (2 *
    L_arm_length * U_arm_length));

var top_angle = 180 - (other_angle * (180/Math.PI));
```



Pitch

Thanks to the leap motion API, there is a pitch function that takes the pitch of your hand, which allowed me to directly map that value to the servo. I fixed the origin so the pitch would be in the middle at 90 by adding 90.

```
pitch_pos = (frame.hands[0].pitch()) * (180/Math.PI) + 90;
```

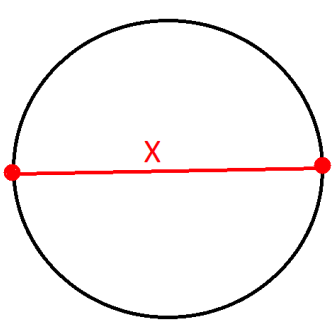
Roll

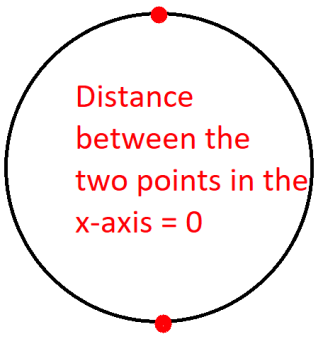
Again, thanks to the leap motion API, there is a roll function that takes the roll of your hand, which allowed me to directly map that value to the servo. Again, I fixed the origin or the roll to my liking.

```
roll_pos = (frame.hands[0].roll()) * (180/Math.PI) + 120;
```

Claw – Grab

The claw was quite tricky to implement. At first I went at it by taking the distance between my thumb and my index finger, specifically the x-axis. I noticed that I had to strictly align those fingers along the x-axis in order for it to work, and if the fingers aren't aligned, then it will act weird. For example, consider taking those two fingers and aligning it along the y-axis, while still doing calculations with the x-axis. While the distance between your two fingers in the y-axis are still wide open, the positions of the two fingers on the x-axis is 0 because they're right on top of each other, causing the claw to grab.

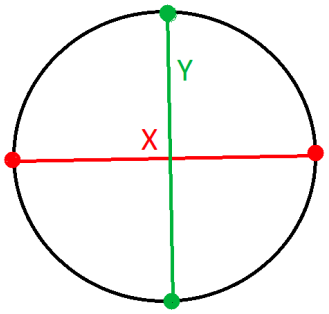




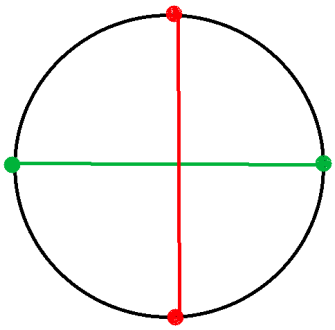
```
var thumb = frame.pointables[0];
var index = frame.pointables[1];
var middle = frame.pointables[2];
var ring = frame.pointables[3];
var pinky = frame.pointables[4];

xfingerDistance = distanceFormula(thumb.tipPosition[0],
    ring.tipPosition[0]);

yfingerDistance = distanceFormula(thumb.tipPosition[1],
    ring.tipPosition[1]);
```

I also tried to take the distance between all 3 points (x,y,z) but that also did not work because if I were to just change one of my fingers positions, such as by moving it further away from the sensor, the distance would change. This is because the leap motion specifically tracks the 3D coordinate of the fingers, not the distance between your fingers. That's why when I take the distance between my fingers, its actually taking the distance between points in the (x,y,z) plane.

My solution around this was to basically OR the X and Y distances together. If you notice, if the distance between two points are 0 on the x-axis, they can be wide open on the y-axis, and vice versa. Taking this into consideration I can OR both values and map it onto values that will work with servos.



Distances:
x = x; x = 0;
y = 0; y = y;



Distances:
x = 0; x = x;
y = y; y = 0;

```
test_pos = xfingerDistance * 2 + yfingerDistance * 2;
test_pos =five.Fn.map(test_pos,180,-180, 0,360) + 70;
```

Known Bugs

- Claw function
    - Even though my solution works perfectly fine as stated above, it doesn't work perfectly. The X OR Y value does move from small values between 0 – 20 (0 is fully open, 180 is fully closed) when I don't specifically minimize the distance between my fingers. This causes the claw to open and close constantly but it doesn't affect the overall functioning. If I were to address this problem in the future, I would probably make more test cases to minimize that problem. I was able to lower it to 0-20 when the same problem occurred with bigger values that make a big difference such as 80-120.

- Leap motion interaction
    - The leap motion is meant to read the palm of your hand, face down over the sensor. This means that when people interact with it strangely, such as turning their hand over where the sensor is reading the top of their hand, it doesn't act right. This causes the robot hand to function a little differently when trying to measure finger tip positions, pitch, roll, etc.. However while this is happening it is still able to read the (x,y,z) position of your hand, therefore the base, shoulder, and elbow would still be able to function fine. If I were to put conditions when a hand interacts with this in a strange way, it would only read the overall position and get rid of the buggy interaction.

Resume/Curriculum Vitae (CV) Blurb

- The Free Motion Robotic Arm utilizes leap motion technology to track hand movements to reflect the same movements onto itself.
- Applied the "Host-Client" method in which a host machine runs Node.js and executes JavaScript logic to communicate with an Arduino Uno to drive the servos of the arm.
- Utilized inverse kinematic equations to determine the joints and angles to provide an efficient and smooth functioning of the robotic arm.

Future work

If I were to continue working on this project, I would use analog feedback in order to be able to "teach" the robot. Each time I move the arm and the servo moves to a new value, I would save the value, and store it in an array. I would then be able to play back the array in a loop for the robotic arm to continuously do what it is taught. If I were to design a case, it would be to hold the wires together for better wire management. I could also have the leap motion sit in one place as well. I don't think creating a custom PCB is necessary because I don't have that many small components. I don't think I would propose to Applied Medical because they build more of the tools, rather than a machine that uses the tools.

References

YouTube video inspiration - https://www.youtube.com/watch?v=5KDY2hj31H8

Leap Motion API –
https://developer.leapmotion.com/documentation/javascript/api/Leap.Frame.html

Helpful Leap Motion and JavaScript example codes and instructions:
https://github.com/DecodedCo/leap-arduino

Helpful Inverse Kinematic Calculations -
https://people.eecs.berkeley.edu/~ug/slide/pipeline/assignments/as10/ik_notes/ik.html

Appendix