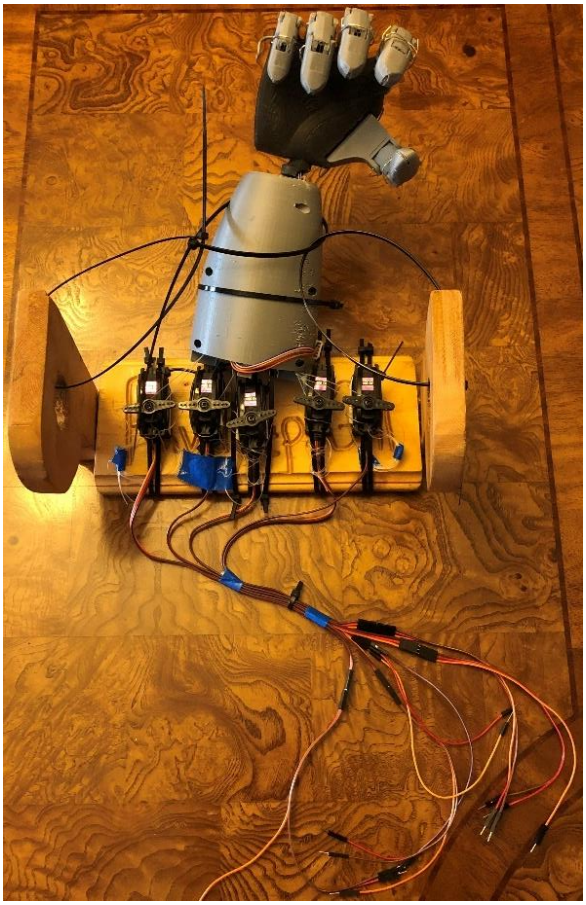
A POLYTECHNIC INSTITUTION

3D-PRINTED BIONIC HAND

by

Pavdeep Gill



Bachelor of Technology in Electronics

Electrical and Computer Engineering Technology

British Columbia Institute of Technology

May 2020

3D-PRINTED BIONIC HAND

by

Pavdeep Gill

Diploma (Automation & Instrumentation), BCIT, 2016

AN INDUSTRY PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIRMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY IN ELECTRONICS

Electrical and computer Engineering Technology

We accept this report as conforming to the required standard

……..

……..

……..

……..

British Colombia Institute of Technology

May 2020

Copyright Pavdeep Gill

ABSTRACT

I was initially thinking of taking ‘Bio-Medical engineering’ instead of ‘Automation & Instrumentation’ as my Diploma of choice at BCIT, so I’ve been interested in making a Bio-Medical project for a while now. The ones that caught my eye right away were Prosthetic/Bionic Hand type projects.

Of course, actually integrating a mechanical bionic prosthetic arm to the nervous system is a lot more complicated, my project was more for a conceptual understanding so I could see how what I learned in the BTECH program transfers over to the Bio-Medical field.

Therefore, my plan was to assemble a 3D-Printed hand and move it via fishing lines connected to DC motors which are then controlled by the Arduino. This project involved ordering the correct parts, searching for plausible designs, programming a microcontroller (Arduino), 3D-printing, assembling each 3D-printed piece together, organizing, spacing each motor and its fishing lines out properly, and fine-tuning / commissioning the end-to-ends. I used Servo motors for the fingers and a Stepper motor for the wrist.

The fingers movement (from closed fist to open fist) via servo motors was successful. The wrist rotation (360°) via the stepper motor was not. The 28BYJ-48 stepper motor doesn’t use PWM signals like the servos do, but instead uses 4 digital signals in order to rotate the DC motor in steps. Its full steps can be controlled via the 4 available pins of the ULN2003 Driver Board. The stepper motor, however, produces a lot less torque than the servo motors and for that reason could not support the weight of the 3D-printed hand.

Next time, I would probably use a 6th servo motor directly mounted (with no attachment) onto the palm/wrist connector rather than a stepper motor. I know I would lose out on 360° rotation capability the stepper motor brings, but the torque produced from the servo motor rotation (capable of only 180°) is more important. Also, the Arduino has 6 PWM signals so PWM signal support should not be a problem.

TABLE OF CONTENTS

ABSTRACT iii
Table of Contents iv
List of Tables vi
Table of Figures vi
INTRODUCTION 1
Project Proposal for 3D-Printed Bionic Hand: 1
3D-Printed Bionic Hand Overview: 2
Actual Hand Anatomy Overview: 3
Actual Wrist Joint 3
Actual Finger Joints 4
DESCRIPTION OF THE PROJECT 5
EXISTING WORK – InMoov Website 6
InMoov Finger Starter Kit 7
SolidWorks (.SLD) Files 9
Approach 10
Export Process for 3D Printing - PLA Filament colour options for 3D-Printing 11
3D-Printed SolidWorks (.SLD) Hand Design PLA Filament Colour Selection 12
3D-Printed Hand Design Stereolithography (.STL) File conversion 13
Palm/Wrist 3D-Print (.STL) File 14
EXPERIMENTAL WORK - Final Config and Cost Breakdown 15
3D-Printed Hand Closed Fist Orientation 15
3D-Printed Hand Final Config 16
3D-Print Hand Design Cost 17
Misc Parts Cost 18
Other Misc Parts 20
Final Estimated Cost 21
EXPERIMENTAL WORK - Tools for Assembly and Production: 22
3D Printer 22
3D Prints Hollow Pieces 23
Hand Tools 24
EXPERIMENTAL WORK - Index Finger, Middle Finger, Ring Finger, Pinky, and Thumb (Servo Motors) 25

Servo Motor Internal Breakdown: 25
Actual MG996R Servo Motor & Attachments 26
Keypad Code: 27
Servo Motor + Switch Code: 28
Switch Schematic: 29
Switch Binary Counter Table: 29
Added Code for PB Counter Control: 30
RIGHT PB being held shown via Arduino Serial Monitor: 32
Final Setup: Servo Motors + Switch + PBs: 33
Final Code: 5 Servo Motors + Switch + PBs: 34
EXPERIMENTAL WORK – Wrist (Stepper Motor) 36
Wrist – 28BYJ-48 5V Stepper Motor + ULN2003 Driver Board 36
28BYJ-48 5V Stepper Motor Calculations: 37
ULN2003A Schematic 37
28BYJ-48 Stepper Motor Code – Wave Driving 40
28BYJ-48 Stepper Motor Code – Full Stepping 42
Wrist – Joystick Schematic and Wiring 43
Wrist – Joystick Code on Arduino UNO 44
Wrist – Joystick Read Results when Joystick is clicked 45
Wrist – Joystick Read Results moving to extremes: All the way UP, DOWN, LEFT, or RIGHT 45
EXPERIMENTAL WORK - Observations and Issues: 47
CONCLUSIONS 48
7 things learned while completing this project: 48
Proposals FOR FURTHER WORK / recommendations 48
REFERENCES 49
APPENDIX 51
TowerPro MG996R Servo Motor Datasheet 51
28BYJ-48 Stepper Motor Datasheet 52
Arduino UNO Pin Layout 53
Notes / Appendix 53

LIST OF TABLES

Table 1 - Switch Binary Counter 29
Table 2 – 28BYJ-48 Stepper Motor's Wave Driving Table 38

TABLE OF FIGURES

Figure 1 – 3D-Printed Bionic Hand Overview (Actual setup is closed fist) 2
Figure 2 – Carpal Bones Example: Abduction, Adduction, Flexion, and Extension 3
Figure 3 – DIP, PIP, and MCP Finger Joints 4
Figure 4 – InMoov Entire Robot Assembly Tutorials 6
Figure 5 – InMoov Finger Starter Kit (see blue box) Tutorial and .STL Files 7
Figure 6 – InMoov Hand & Forearm Tutorial and downloadable Files (.STL) 8
Figure 7 - Left/Right Hand + Wrist Pieces 8
Figure 8 – .SLD Files for 3D-Printed Hand 9
Figure 9 – .SLD File Measurement of Palm/Wrist Connector 10
Figure 10 – Polylactic Acid (PLA) Material for 3D Printing 11
Figure 11 – SolidWorks (.SLD) Colour Design 12
Figure 12 – SolidWorks (.SLD) 3D-Print Export into Stereolithography (.STL) 13
Figure 13 – Modified Palm/Wrist Connector (.STL) File 14
Figure 14 – 3D-Printed Bionic Hand Overview (Open and Closed Fist Orientations) 15
Figure 15 – Final 3D-Printed Hand Config 16
Figure 16 – 3D-Print of Hand Design 17
Figure 17 – Misc Parts 1: Stepper Motor, Portable Chargers (5V Source), Breadboards, Breadboard Jumpers 18
Figure 18 – Misc Parts 2: Servo Motors, Keypad, Hot Glue, Gorilla Glue, Paper Clips, and Zip ties 19
Figure 19 – Other Misc Parts: Fishing Line & Galvanized steel wire 20
Figure 20 – Ultimaker 3D Printer 22
Figure 21 - 3D-Printed hollow pieces ripped out 23
Figure 22 – Wire Cutter, Needle Nose Plier, Xacto Knife, Ruler, and Scissors 24
Figure 23 – Servo Motor Internal Breakdown 25
Figure 24 – Actual MG996R Servo Motor & Attachments 26
Figure 25 – Keypad Test Code 27
Figure 26 - Servo Motor + Switch Code 28
Figure 27 - Two out of the Four Switch Module's Schematic (Two-Bit Binary Counter) 29
Figure 28 – Servo Motor + Switch +LEFT & RIGHT PB's Code 31
Figure 29 - PB Schematic 31
Figure 30 - Serial Plotter of RIGHT PB being held 32
Figure 31 – Final Servo Motors + Switch + PBs Setup 33
Figure 32 - Final Code for 5 Servo Motors + Switch + PBs 35
Figure 33 - 5V Stepper Motor 28BYJ-48 and ULN2003A Driver Board 36
Figure 34 - ULN2003A's Schematic 37

Figure 35 - 28BYJ-48 Stepper Motor Coil Labels 38
Figure 36 – ULN2003 Driver Board slowed down to show A, B, C, and D Bits / Coils activating one at a time (Red LED) 39
Figure 37 - Stepper Motor – Wave Driving Code 40
Figure 38 – Stepper Motor – Wave Driving (one phase on at a time and 11.25 Step Angle) 41
Figure 39 – Stepper Motor – Full Stepping (Two phases on at a time and 11.25 Step Angle) 42
Figure 40 – Arduino UNO and Joystick Wiring 43
Figure 41 - Arduino UNO and Joystick Actual Wiring 43
Figure 42 - Arduino UNO read code for joystick 44
Figure 43 - Arduino UNO code result for when Joystick is clicked 45
Figure 44 - Arduino UNO code result when Joystick moved All the Way Up 45
Figure 45 - Arduino UNO code result when Joystick moved All the Way Down 46
Figure 46 - Arduino UNO code result when Joystick moved All the Way Left 46
Figure 47 - Arduino UNO code result when Joystick moved All the Way Right 46

INTRODUCTION

Project Proposal for 3D-Printed Bionic Hand:

Hi Gino, I was thinking of doing the project idea that is listed below:

ELEX 8300 Industry Project Proposal

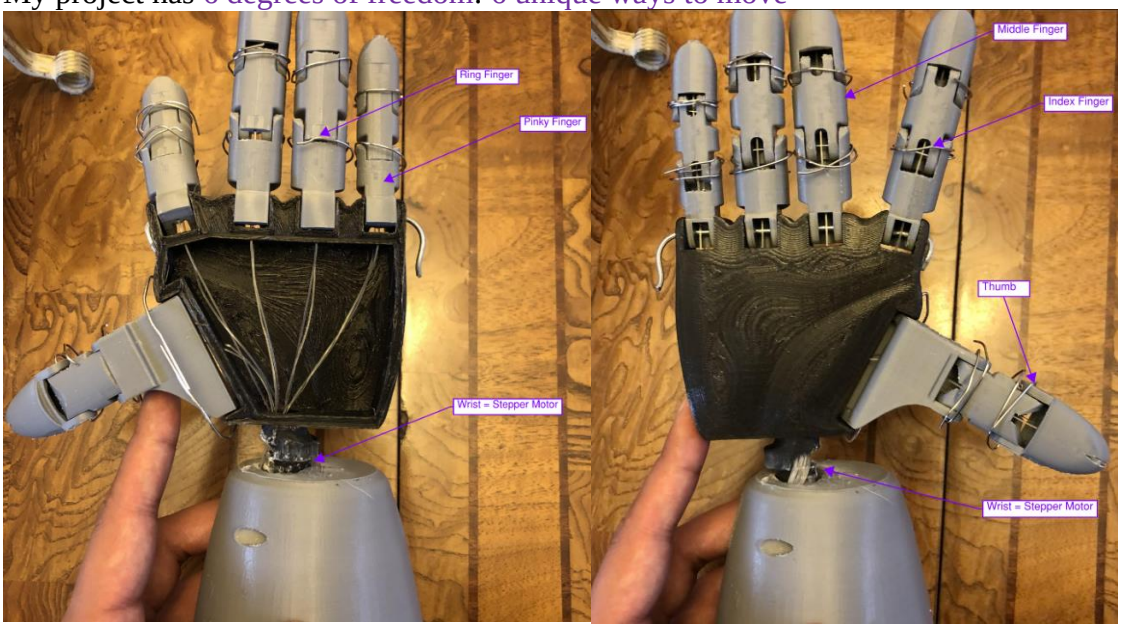
Title of your project: 3D-Printed Bionic Hand (initially was Prosthetic/Bionic Hand)
Your cell phone number: 604-374-0202
Name of your Faculty Advisor: Gino Carrese
Name of the company you work for: Currently not working
Will the project be for your employer or independent: Independent

Project Description: I want to make a model on SolidWorks software and then create an actual 3D- printed robotic hand from that model that will move its fingers according to programmed functions.

Parts Description: Use Ultimaker 3D printer in MediaWorks room, make 3D-printed parts for hand, string/fishing line, screws, use servo motors, wires, Arduino, and laptop

3D-Printed Bionic Hand Overview:

My project has 6 degrees of freedom: 6 unique ways to move



<u>Figure 1</u> – 3D-Printed Bionic Hand Overview (Actual setup is closed fist)

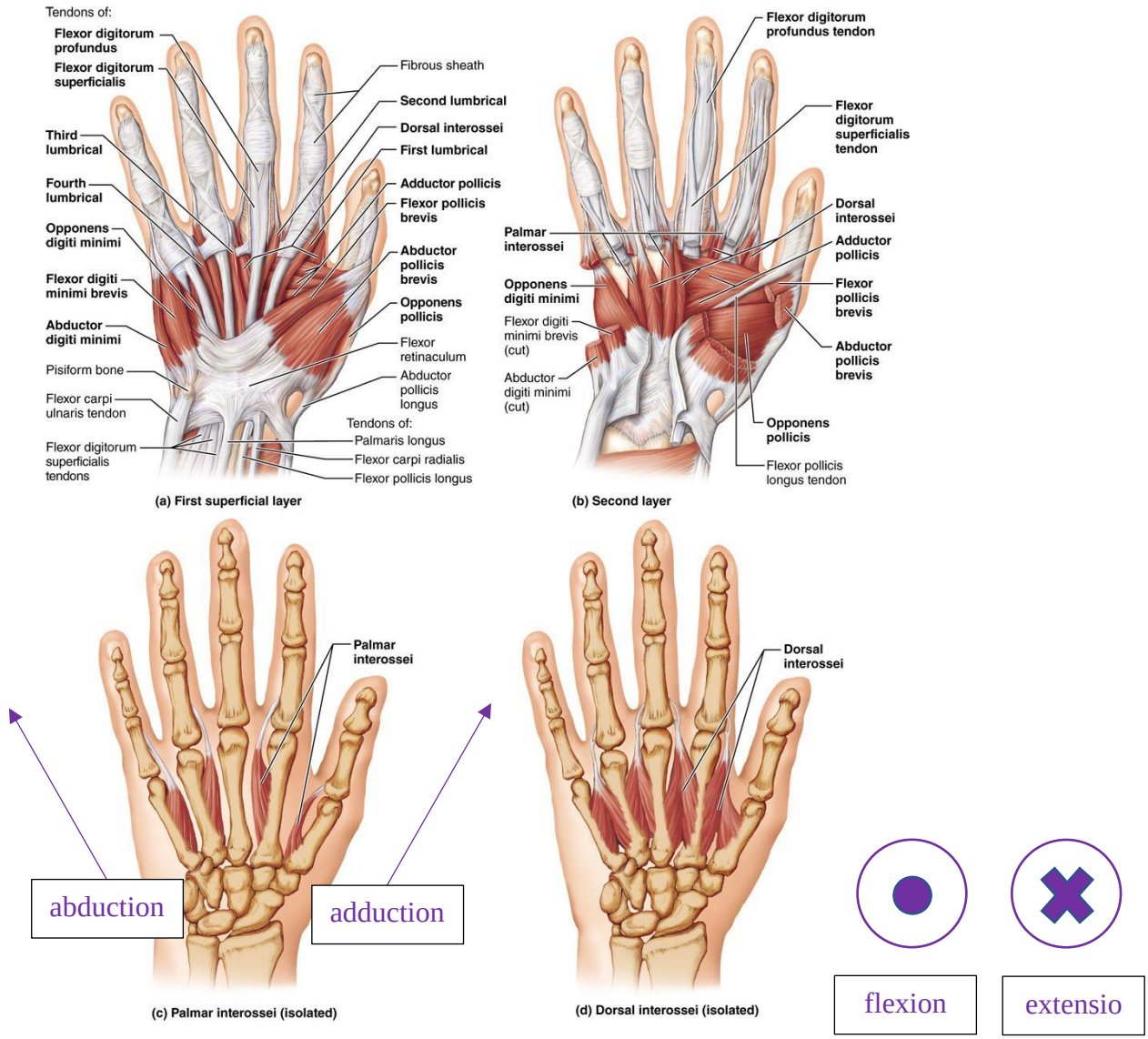
5 Fingers + Wrist: Index Finger, Middle Finger, Ring Finger, Pinky, Thumb, and Wrist

- Each finger can move up (would move down if I had open fist orientation) = (via 5 **Servo Motors**)
- Wrist can rotate 360° (via 1 **Stepper Motor**)

Actual Hand Anatomy Overview:

An actual hand has a lot more degrees of freedom because the muscle fibres are very small compared to servo motors.

Actual Wrist Joint



<u>Figure 2</u> – Carpal Bones Example: Abduction, Adduction, Flexion, and Extension

- An actual wrist joint can usually rotate about 180°, and lean to left (abduction), lean to right (adduction), lean forward (flexion), lean backward (extension)

Actual Finger Joints

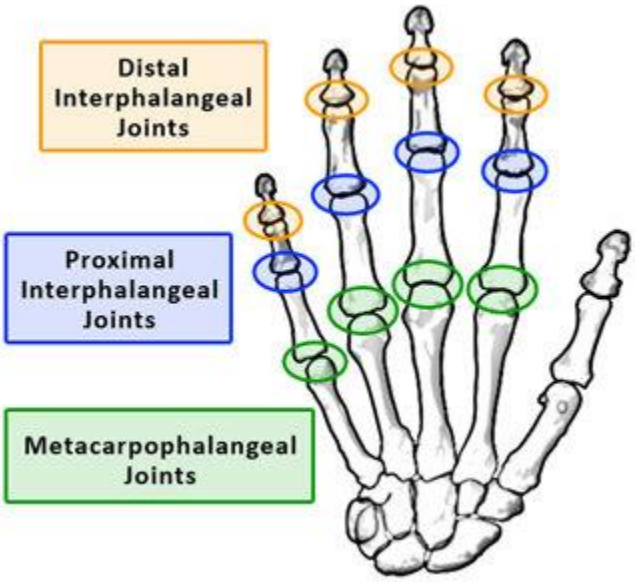


Figure 3 – DIP, PIP, and MCP Finger Joints

Each finger has 3 joints:

- Metacarpophalangeal joint (MCP) – the joint at the base of the finger. The MCP joint can rotate a little, lean to left (abduction), lean to right (adduction), lean forward (flexion), lean backward (extension), and a combination of flexion, extension, adduction and abduction which is called (circumduction).
- Distal interphalangeal joint (DIP) – the joint closest to the fingertip. The DIP joints can only lean forward (flexion) and lean backward (extension).
- Proximal interphalangeal joint (PIP) – the joint in the middle of the finger. The PIP joints can only lean forward (flexion) and lean backward (extension).

DESCRIPTION OF THE PROJECT

I watched a lot of YouTube videos to plan out and reassure myself that I could bring this project into fruition. I then looked for DC Motors. I eventually stumbled upon Servo and Stepper DC Motors in my research. After understanding the functionality, I found those to be the right DC Motors for the job and placed an order. I also ordered several other parts such as the Arduino Uno, fishing line, hot glue, keypads, joysticks, and portable chargers. I then attended a 3D-Printing workshop hosted by Richard Te and Ken in the MediaWorks room SE14-250 to prepare (BCIT, 2020). Lastly, I practiced coding on the Arduino and went over the Arduino reference library (Arduino, 2020).

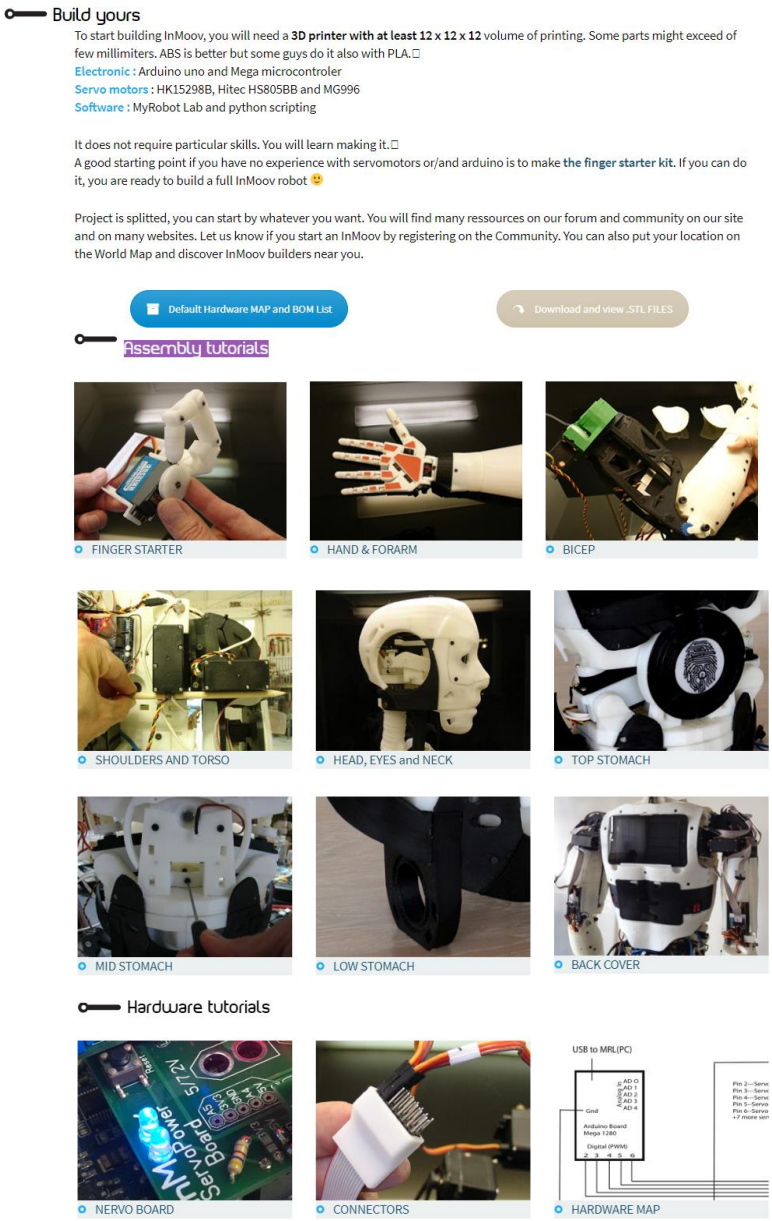
I then searched for SolidWorks hand designs on the web through forums, tinkering websites, YouTube, etc. Since I didn't get much SolidWorks design experience in school, I decided to look for help on the web. I found plenty of websites with .STL or .SDL files for hand model designs such as InMoov, GrabCAD, YouTube Channels, etc. STL is just a format (often used in 3D Printing) that can be saved from the original SolidWorks design (.SDL).

I then printed all the 3D components and compared with the SolidWork models to confirm which finger was which. Then I got rid of the hollow filament and assembled the pieces together, ensuring that they slotted well. As I was combining pieces together, I ran fishing line through the holes of each piece (if there was filament blocking the path, I used an Xacto knife to clean it up). For example, each finger was made up of 3 pieces, that slotted together at the joints (see *Figure 3*). Then I kept each piece in place with paper clips and zip ties.

After, assembly was complete, I moved onto testing and commissioning, where I found some parts to be faulty and others to lack capability (torque). I then readjusted the code, some parts, and proceeded on with commissioning and testing. I documented along the way and edited / finalized at the end.

EXISTING WORK – INMOOV WEBSITE

One website that caught my interest the most is called InMoov, it shows tutorials for an entire 3D Printed Robot. It also provides free designs (.STL) files for pieces of the robot (Langevin, InMoov body parts library, 2020).

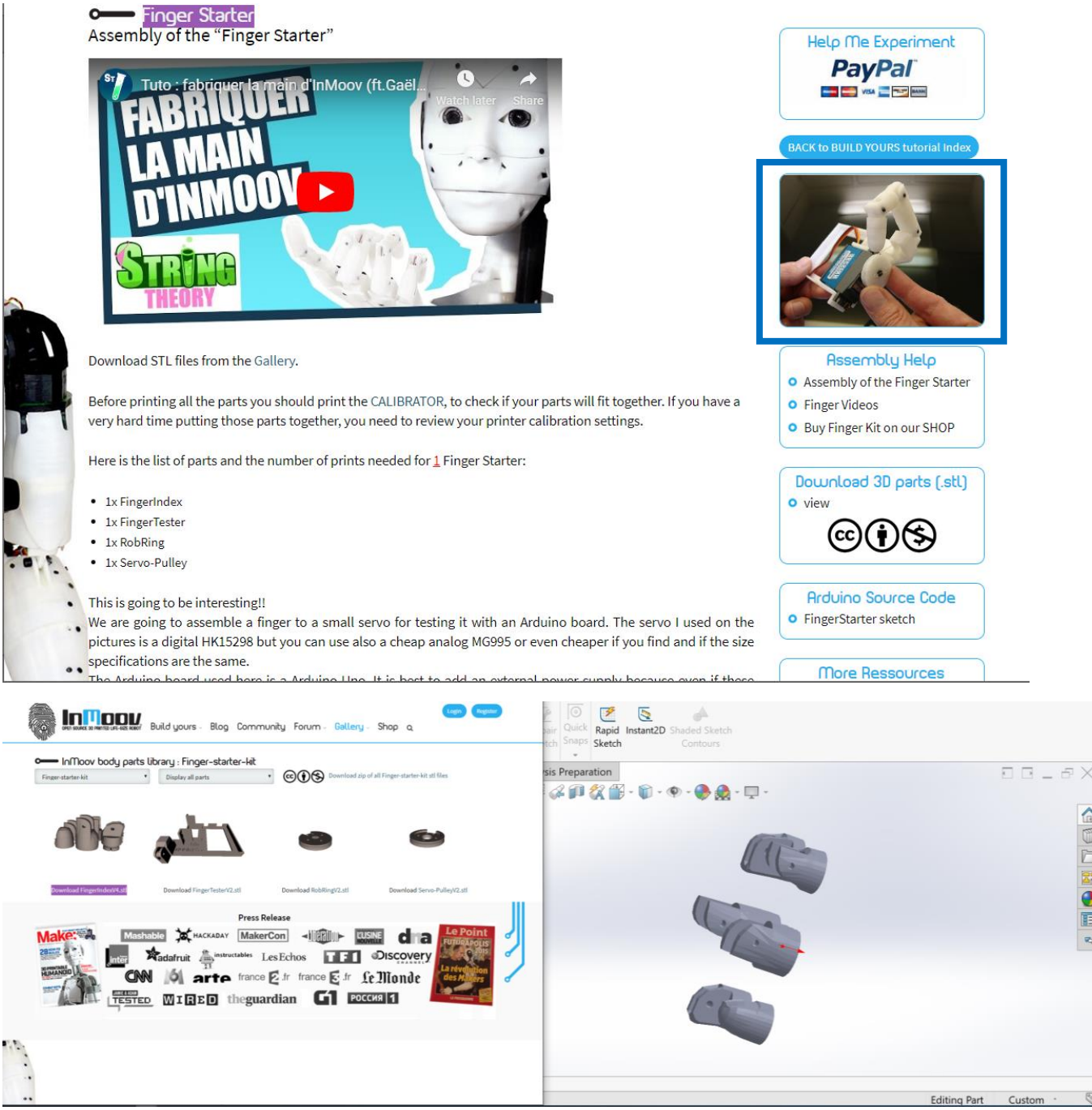


(Langevin, Build yours, 2020)

<u>Figure 4</u> – InMoov Entire Robot Assembly Tutorials

- Assembly tutorials available on InMoov's website (BUILD YOURS): 'Finger Starter' and 'Hand & Forearm'

InMoov Finger Starter Kit



(Langevin, Build yours, 2020)

Figure 5 – InMoov Finger Starter Kit (see blue box) Tutorial and .STL Files

- Tutorial is compromised of pictures and captions and there is also a 14-minute subtitled YouTube video (in French).

Hand and Forarm

Download STL files from the Gallery.

Before printing all the parts you should print the CALIBRATOR, to check if your parts will fit together. If you have a very hard time putting those parts together, you need to review your printer calibration settings.

Here is the list of parts and the number of prints needed for 1 right hand and forarm:

- 1x Thumb
- 1x Index
- 1x Majeure
- 1x RingFinger
- 1x Auriculaire (Pinky)
- 1x Bolt_entretoise
- 1x Wristlarge
- 1x Wriststmall
- 1x topsurface
- 1x coverfinger
- 1x robcap3
- 1x robpart2
- 1x robpart3
- 1x robpart4
- 1x robpart5
- 1x ElbowShaftGear (if you built the bicep)

Here is the list of parts and the number of prints needed for 1 right wrist:

- 1x rotawrist2
- 1x rotawrist1
- 1x rotawrist3
- 1x WristGears
- 1x CableHolderWrist

Figure 6 – InMoov Hand & Forearm Tutorial and downloadable Files (.STL)

- I focused mainly on just the hand and wrist pieces (ignored arm).

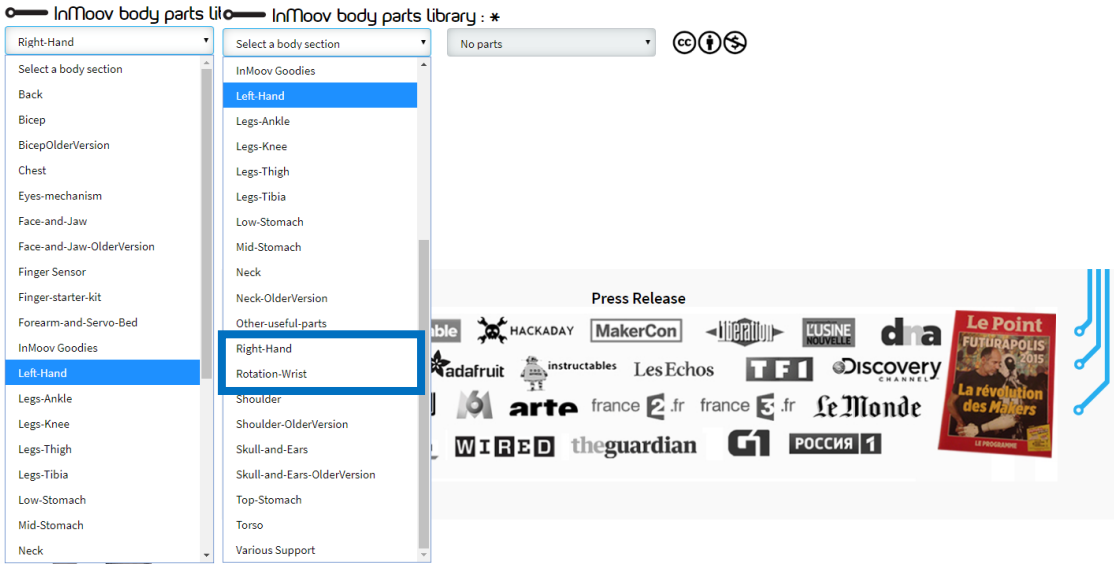


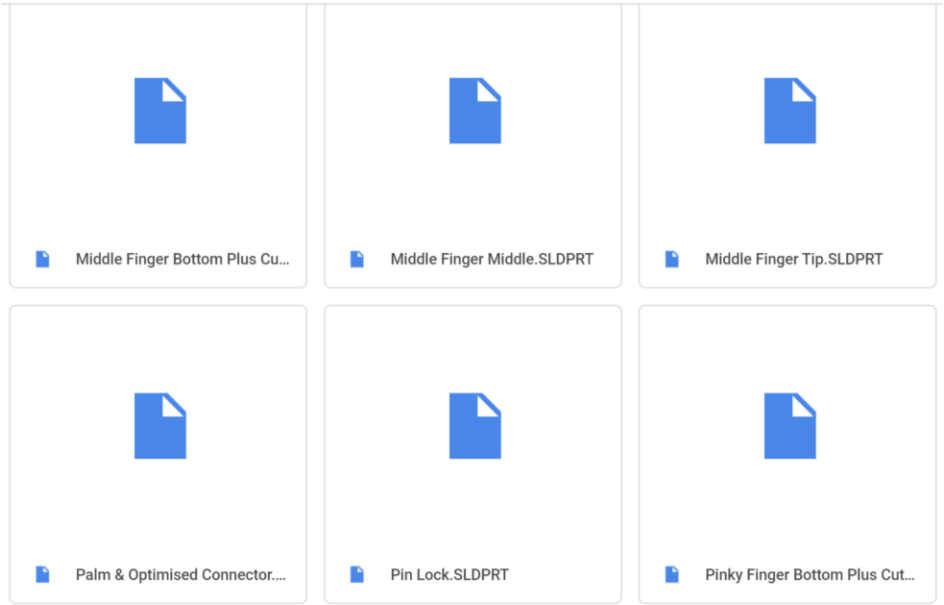
Figure 7 - Left/Right Hand + Wrist Pieces

(Langevin, InMoov body parts library, 2020)

- I just focused on the hand part because that is what I am most interested in.

SolidWorks (.SLD) Files

I didn't want to use gears or drill the motors onto threaded filament, so I searched for hand designs similar but slightly different than the InMoov project and ended up using the following design:



(Solidworks Models, 2017)

Figure 8 – .SLD Files for 3D-Printed Hand

APPROACH

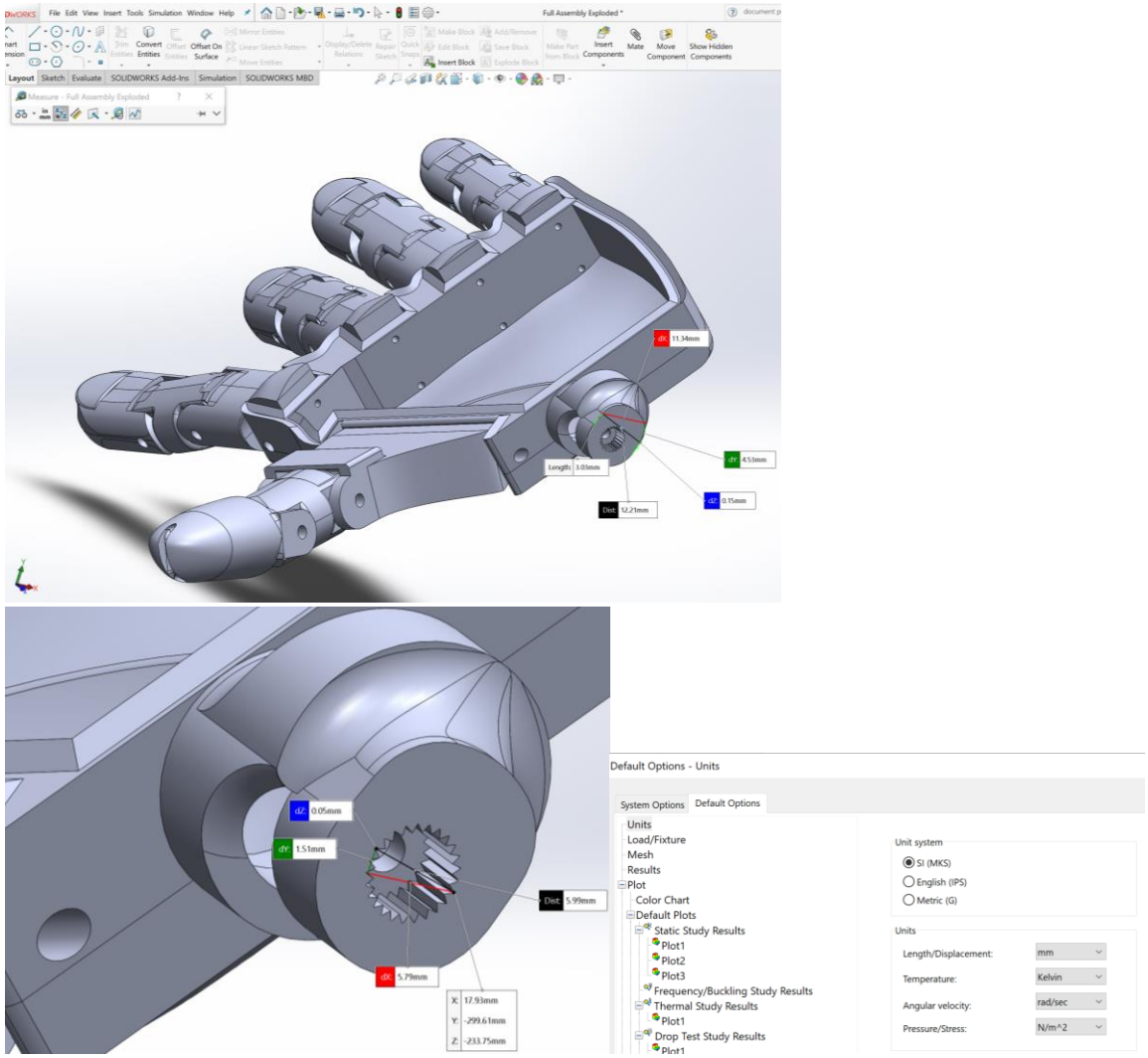


Figure 9 – .SLD File Measurement of Palm/Wrist Connector

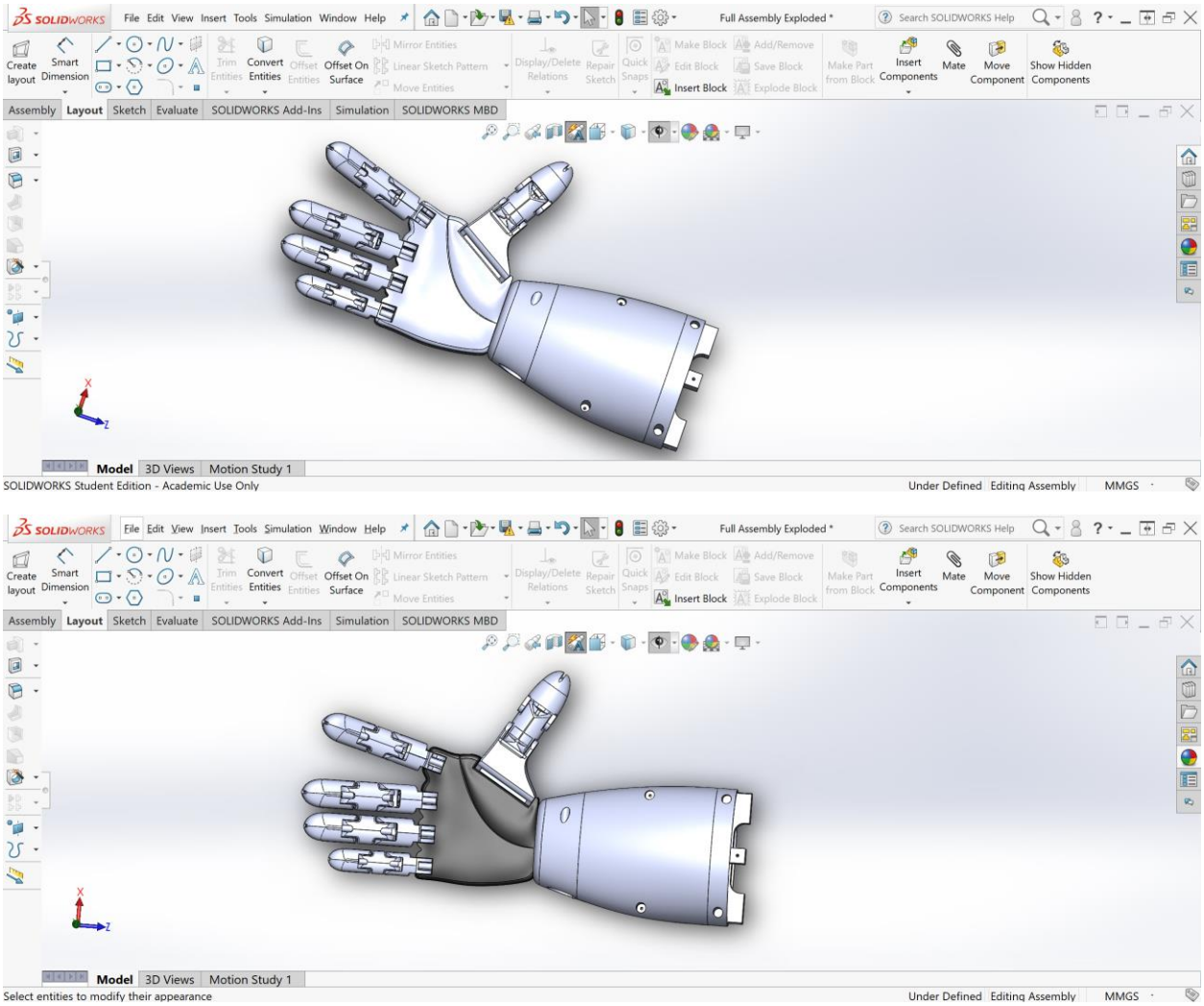
- I made some measurements (SI Units: 12.21 mm & 5.99 mm) in SOLIDWORKS of the hand.
- I then made some slight mods for my parts (MG996R Servo's and 28BYJ-48 Stepper motors).

Export Process for 3D Printing - PLA Filament colour options for 3D-Printing

Figure 10 – Polylactic Acid (PLA) Material for 3D Printing

- The MediaWorks room SE14-250 has White, Red, Yellow, Green, Silver and Black PLA filament. I decided to print majority of the hand in silver filament but the palm in black filament to add some colour difference.

3D-Printed SolidWorks (.SLD) Hand Design PLA Filament Colour Selection



<u>Figure 11</u> – SolidWorks (.SLD) Colour Design

- Mainly went with Silver PLA Filament
- I added black filament to Palm / Wrist for a little variety.

3D-Printed Hand Design Stereolithography (.STL) File conversion

I saved the Solidworks file as a .STL file by going to File → Save As on Solidworks. STL is just a format (often used in 3D Printing) that can be saved from the original SolidWorks (.SDL) design.

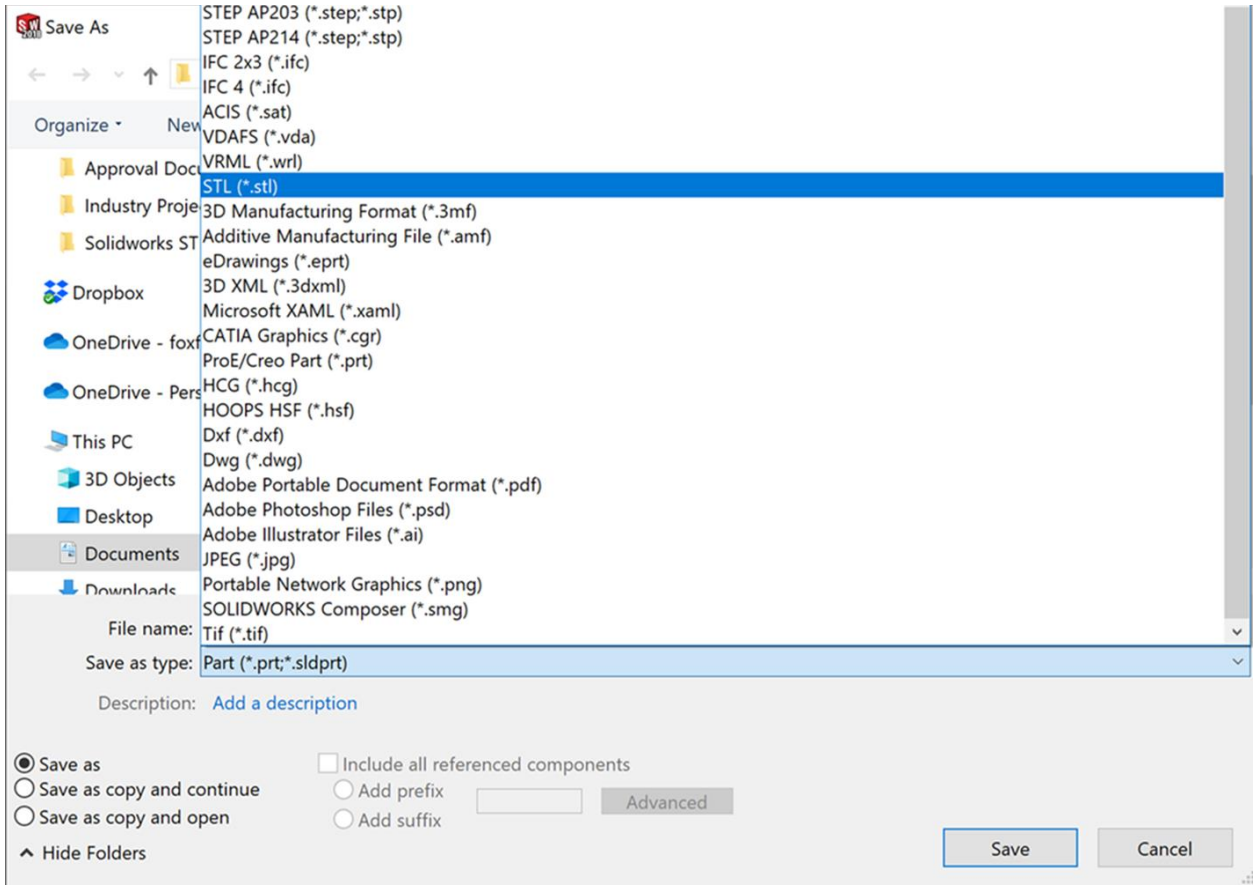


Figure 12 – SolidWorks (.SLD) 3D-Print Export into Stereolithography (.STL)

- STL (an abbreviation of "stereolithography") is a file format native to the stereolithography CAD software created by 3D Systems. STL has several after-the-fact backronyms such as "Standard Triangle Language" and "Standard Tessellation Language".

Palm/Wrist 3D-Print (.STL) File

STL files describe only the surface geometry of a three-dimensional object without any representation of color, texture, or other common CAD model attributes. Therefore, I told Richard Te what color filament I want myself.

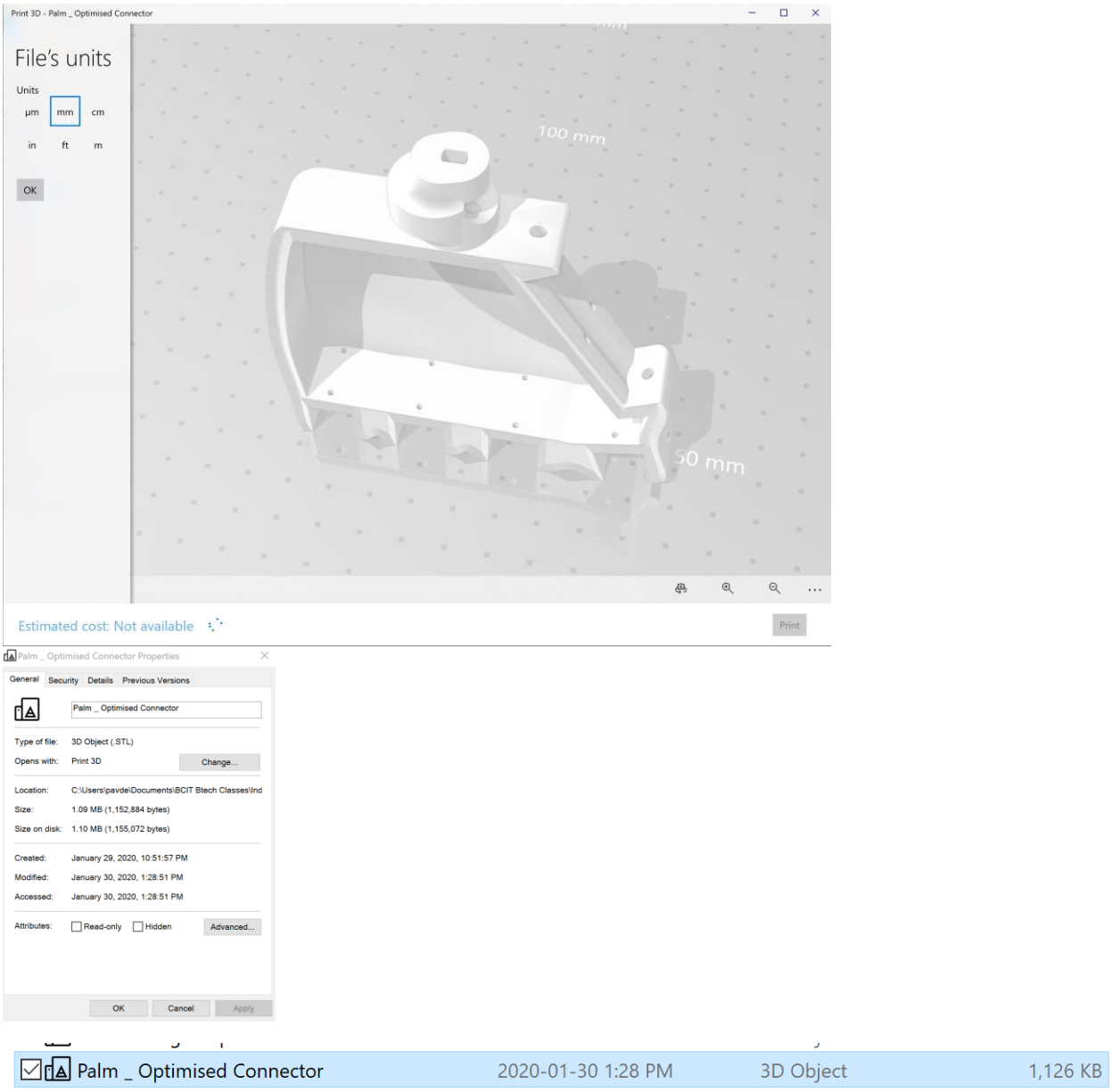


Figure 13 – Modified Palm/Wrist Connector (.STL) File

- I converted the .SLD files (shown in previous figure) to .STL files (for printing).

EXPERIMENTAL WORK - FINAL CONFIG AND COST BREAKDOWN

3D-Printed Hand Closed Fist Orientation

As shown on the **Actual Hand Anatomy Overview** section, there are 3 main joints (see *Figure 3*). Therefore the 3 joints, in the 3D-printed hand model, are designed to slot together nicely. I then passed a paper clip through the middle slot to connect the finger components together, and lastly bent / wrapped the paper clip around it so they would stay joined - while making sure the fishing line stayed passing through each component.

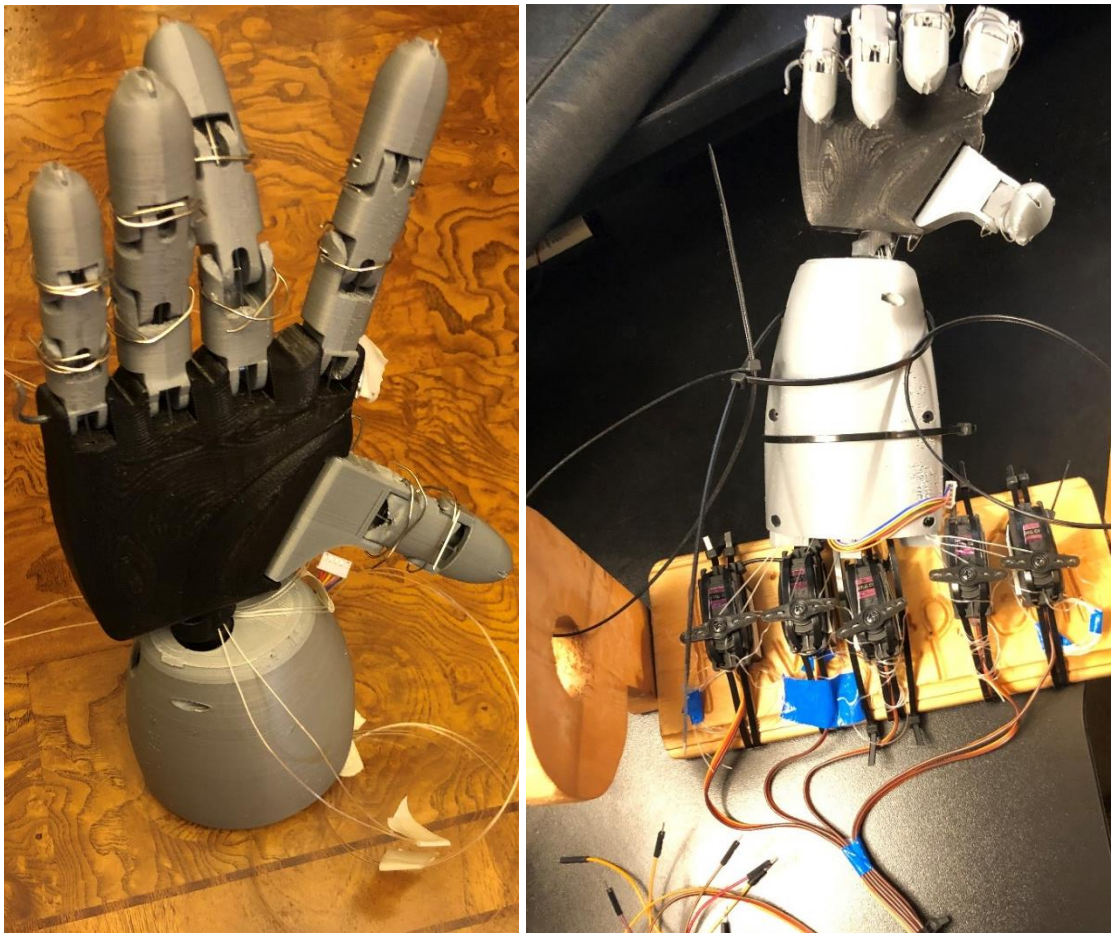


Figure 14 – 3D-Printed Bionic Hand Overview (Open and Closed Fist Orientations)

- Final Orientation chosen was closed fist
- Final Config is shown on Right Hand Side (RHS)

3D-Printed Hand Final Config

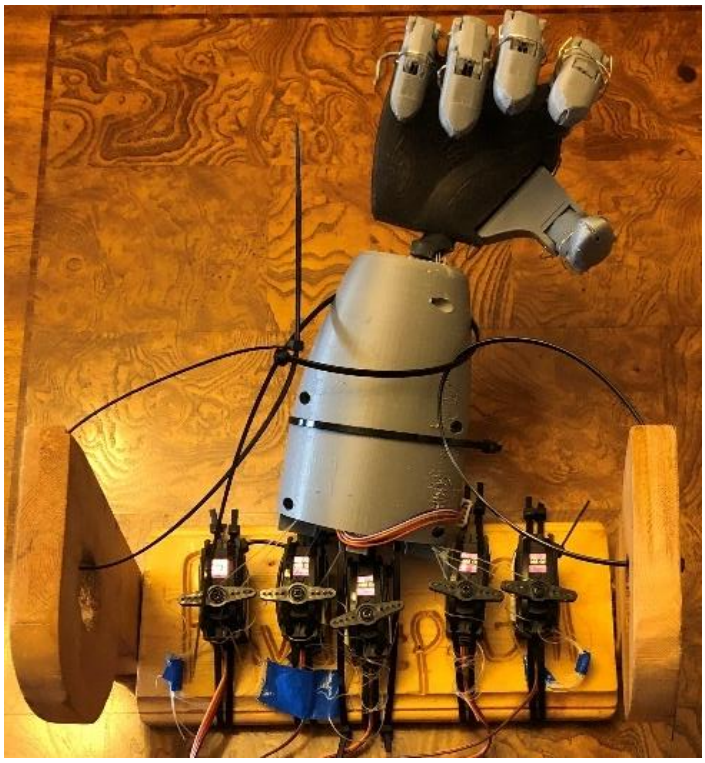


Figure 15 – Final 3D-Printed Hand Config

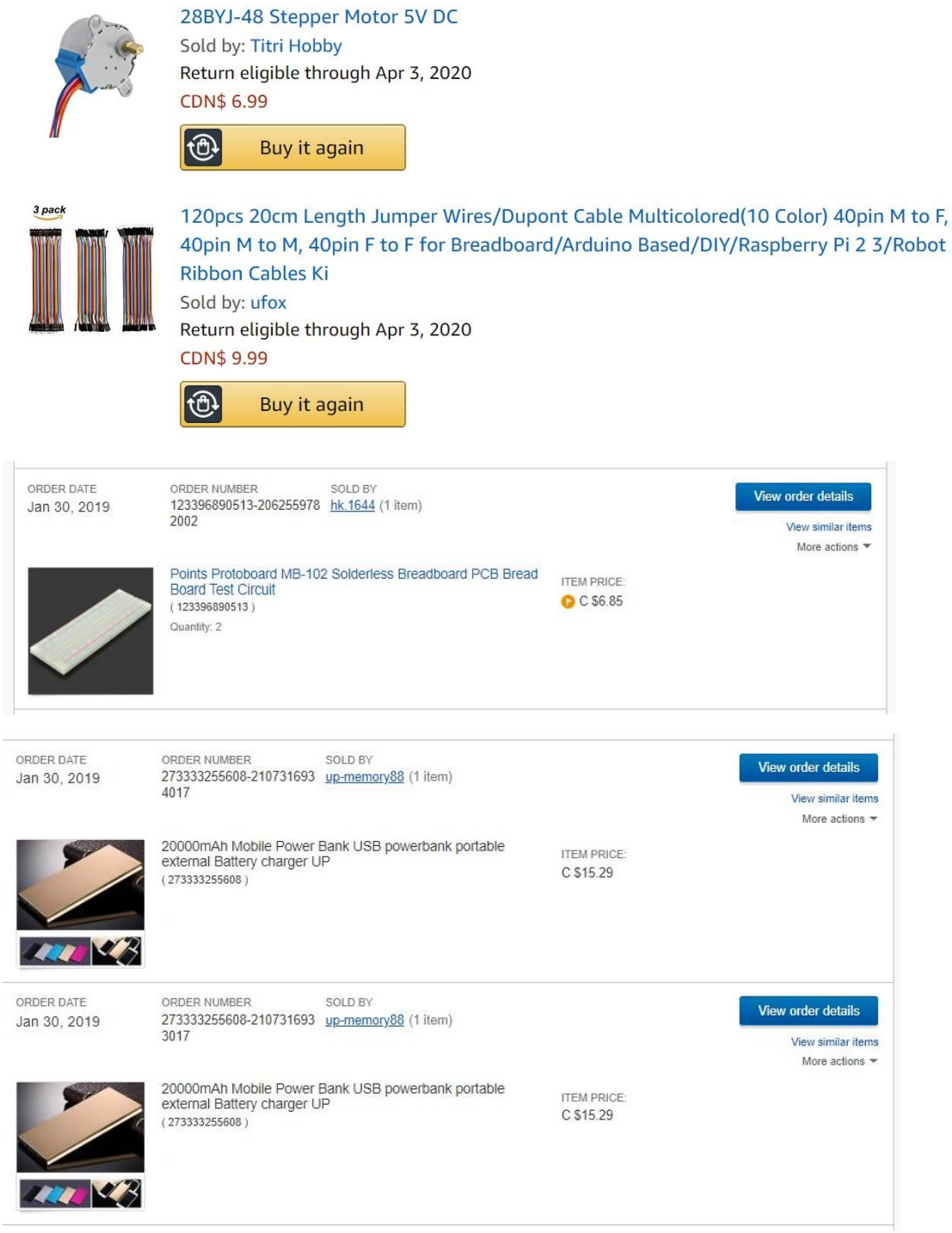
- This design is similar to InMoov's but without gears / motors mounted in palm (I mounted on a piece of wood)

3D-Print Hand Design Cost

<u>Figure 16</u> – 3D-Print of Hand Design

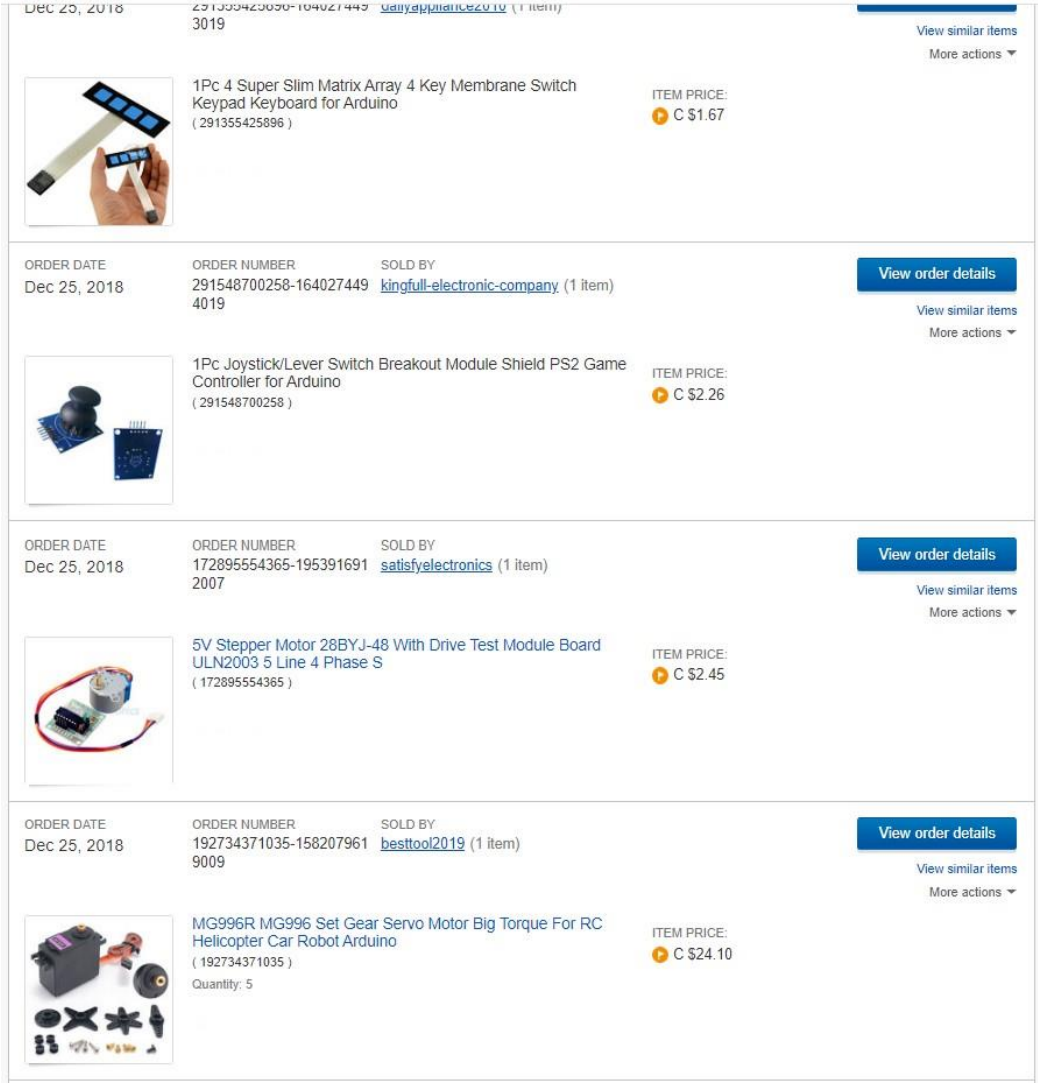
- The 3D-Printer prints everything from the base up in PLA filament
  Cost was 357 grams * 0.10 ¢ / gram = **$35.70**

Misc Parts Cost



<u>Figure 17</u> – Misc Parts 1: Stepper Motor, Portable Chargers (5V Source), Breadboards, Breadboard Jumpers

- $6.99 + $9.99 + $6.85 + $15.29 + $15.29 = **$54.41**



<u>Figure 18</u> – Misc Parts 2: Servo Motors, Keypad, Hot Glue, Gorilla Glue, Paper Clips, and Zip ties

- Misc 2: $24.10 + $1.67 + $2.26 + $2.45 + (~$12) = **$30.48 (~$42.48)**
- Arduino Uno = **$35.00**
- A third portable charger = **$15.00**

Other Misc Parts

<u>Figure 19</u> – Other Misc Parts: Fishing Line & Galvanized steel wire

- Around **$10.00 - $15.00**

Final Estimated Cost

To recreate this project from scratch (no parts), it would cost around **$185.00**

- Around $180.59 - $197.59

However, I had some of the parts already: Arduin UNO, a portable charger, Hot Glue, Gorilla Glue, Paper Clips, and Zip ties:

- So, it probably ended up costing me around: **$123**

If I didn't order extra parts (breadboards, jumpers, and extra stepper motor) then the price could have been cutdown to about **$100**

Note: The extra stepper motor was used to confirm issue of hand's weight not being able to be supported via Stepper Motor.

In reality, since I actually didn't get to use the stepper motors and accessories in my project, I could subtract another 8 bucks or so. So, I would say my **final cost for** the **project is** actually about **$82.00** - which is reasonable.

EXPERIMENTAL WORK - TOOLS FOR ASSEMBLY AND PRODUCTION:

3D Printer

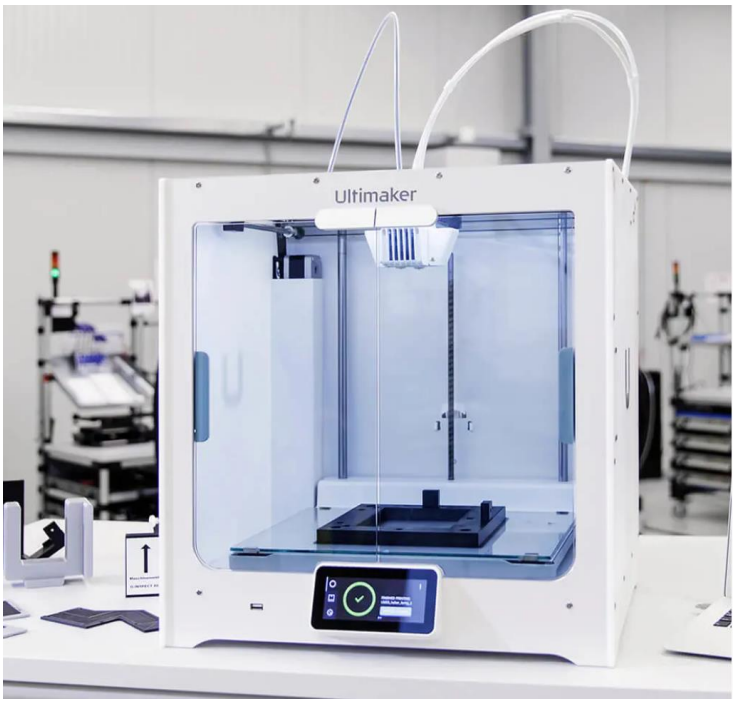


Figure 20 – Ultimaker 3D Printer

3D Prints Hollow Pieces

A lot of pieces are not completely made up of solid filament, but instead consist of a hollow piece inside. The hollow parts of the components are not apart of the actual design and can be ripped off (I used a Needle Nose Plier). The hollow pieces are still there because the 3D printer technically must still print everything continuously from the base up, so some gaps may come out as hollow when produced.

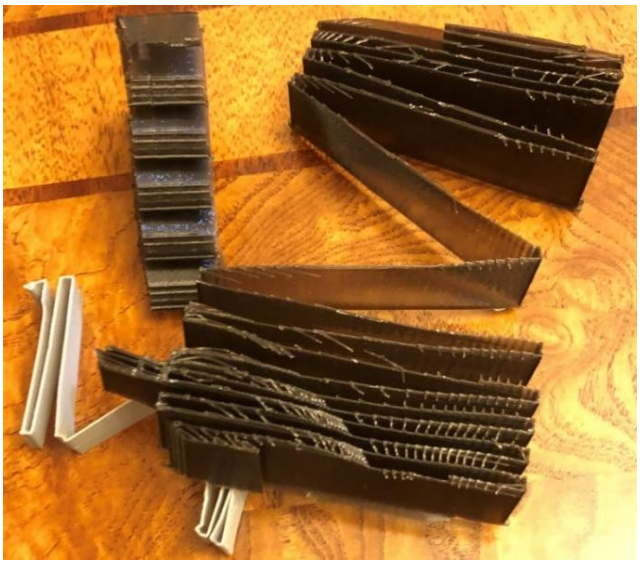


Figure 21 - 3D-Printed hollow pieces ripped out

Hand Tools

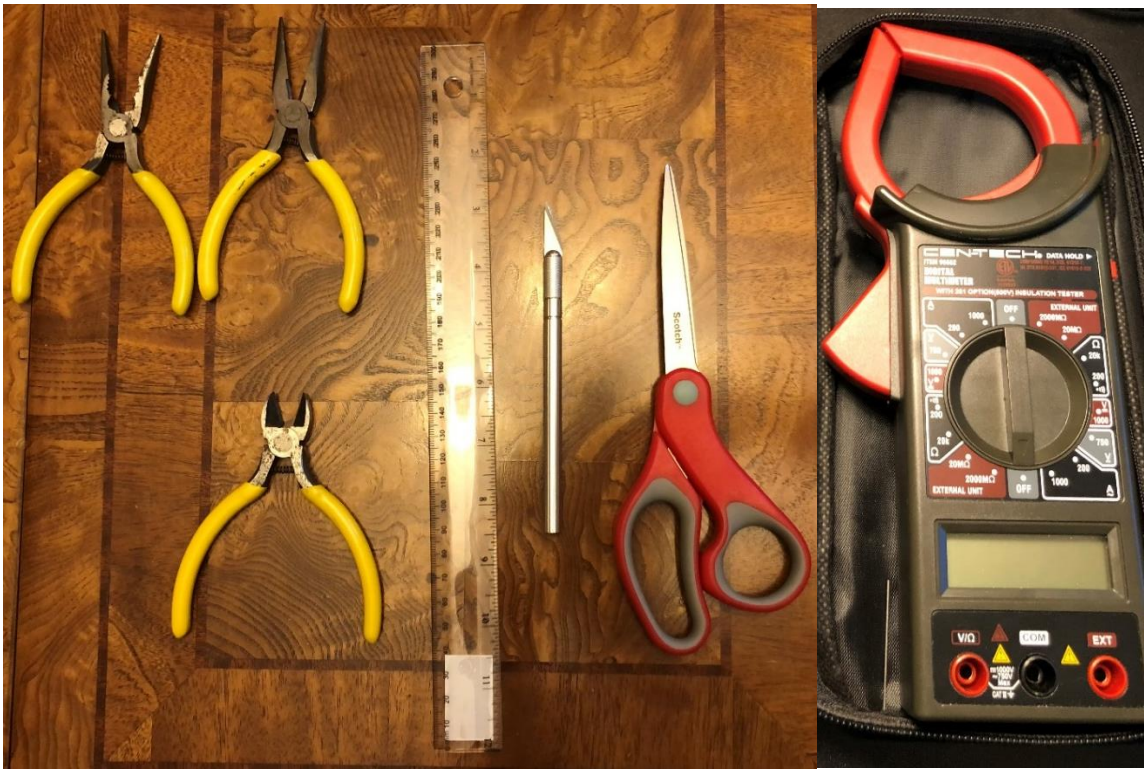


Figure 22 – Wire Cutter, Needle Nose Plier, Xacto Knife, Ruler, and Scissors

- Wire Cutter – used to snip paper clips (wires I already used assorted jumper sizes)
- Needle Nose Plier – used to pull out hollow/unwanted pieces
- Xacto Knife – used to make sure joint holes were clear
- Flathead Screwdriver – to attach servomotor head attachment
- Hot Glue Gun – to glue the stepper motor to the palm/wrist
- Multimeter

EXPERIMENTAL WORK - INDEX FINGER, MIDDLE FINGER, RING FINGER, PINKY, AND THUMB (SERVO MOTORS)

Servo Motor Internal Breakdown:

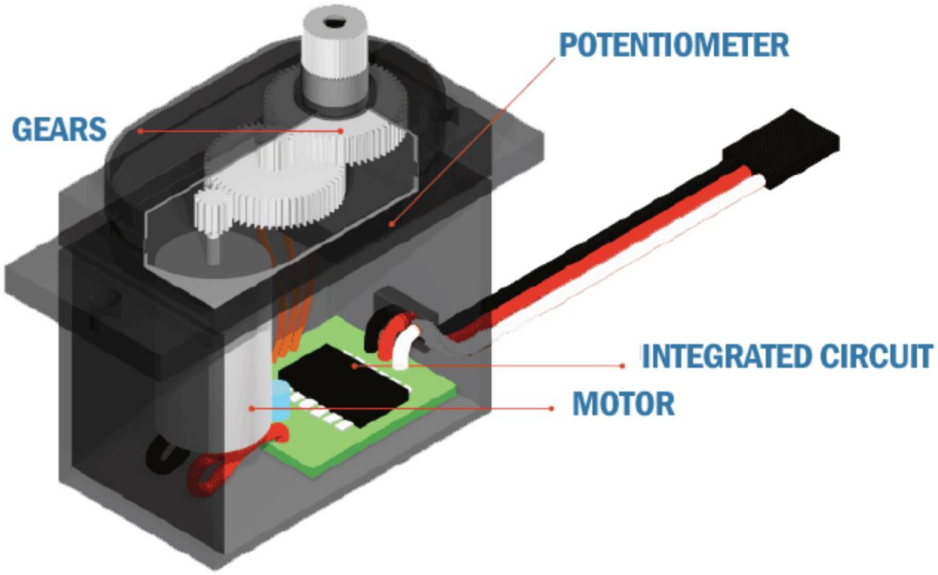


Figure 23 – Servo Motor Internal Breakdown

(Control a 'LOT' of Servo Motors using a Joystick, Arduino and PCA9685 PWM Module, 2016)

- I used MG996R Servo Motors for my project
- It's cable connector is made up of 3 coloured signal wires bundled together, which are actually not Black (GND), Red (5V), and White (PWM Signal), but instead are coloured Brown (GND), Red (5V), and Orange (PWM Signal).

Actual MG996R Servo Motor & Attachments

A servo motor is typically compromised of gears, some type of limit switch, a shaft, a potentiometer, and three wires for VCC, GND, and PWM. I used the TowerPro MG996R, which is the upgraded version of the very popular MG995 servo. I was initially going to use a 4-digit keypad to program 4 operations (4 different finger commands) to the servo, but the keypad ended up being a faulty piece. Therefore, I decided to go with two switches instead, and to just use them as a two-bit binary counter (0,1,2,3) for the 4 modes of operation.

<u>Figure 24</u> – Actual MG996R Servo Motor & Attachments

- 5V Servo Motor (High Torque)
- Weight = 55 grams
- Stall torque: 9.4 kg/cm (4.8v)
- Dimension: 40.7 mm × 19.7 mm × 42.9 mm
- Running Current 500 mA - 900 mA (6V)

(MG996R Tower-Pro, n.d.)

Keypad Code:

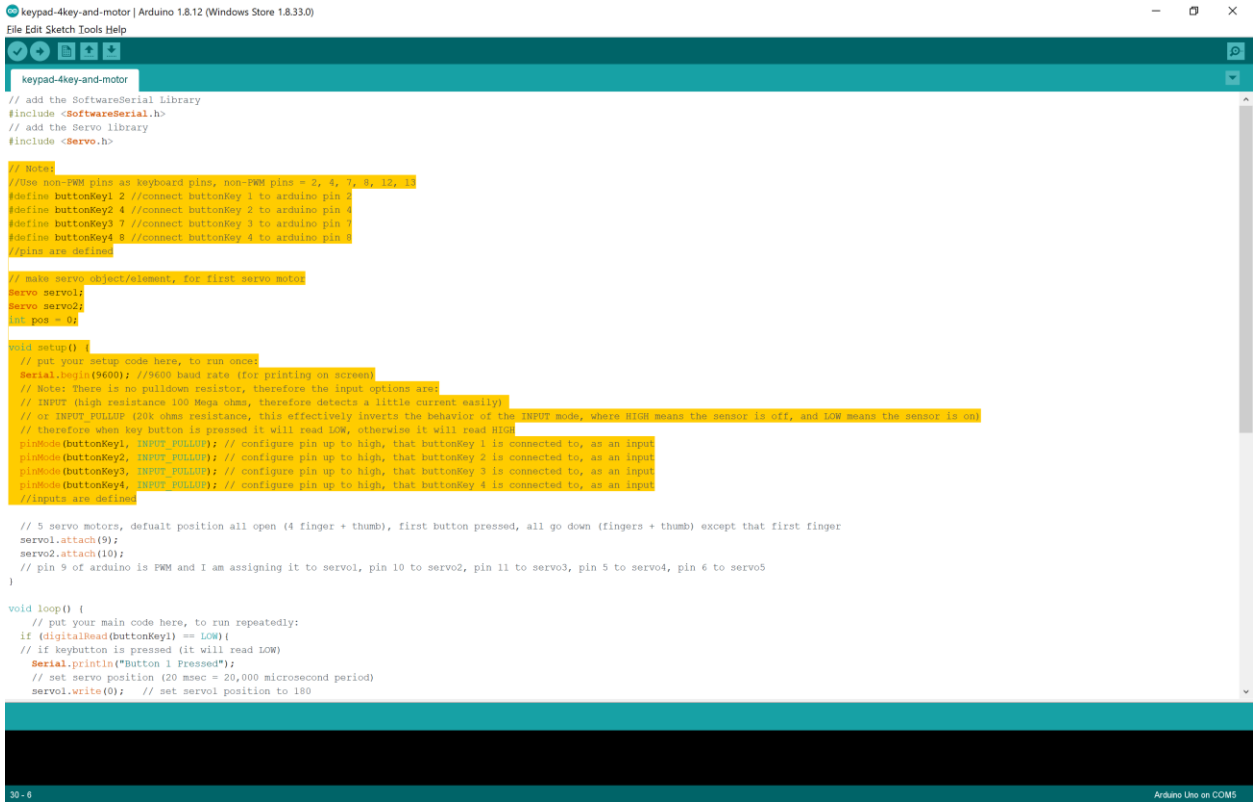


Figure 25 – Keypad Test Code

- Highlighted shows Buttons 1 - 4 (buttonKey1 – buttonKey4) are mapped to Arduino I/P pin's 2, 4, 7, and 8
- Tested 4 button actions with 2 servos attached to Arduino's PWM Outputs 9 - 10

Note: Keypad was not used, went with switch instead because some of the keypad buttons were not being read correctly (poorly manufactured).

Servo Motor + Switch Code:

switch_binary_counter-and-motor.ino | Arduino 1.8.12 (Windows Store 1.8.33.0)

File Edit Sketch Tools Help

switch_binary_counter-and-motor.ino

```
// add the SoftwareSerial Library
#include <SoftwareSerial.h>
// add the Servo library
#include <Servo.h>

// Note:
//Use non-PWM pins as keyboard pins, non-PWM pins = 2, 4, 7, 8, 12, 13
#define buttonKey1 2 //connect buttonKey 1 to arduino pin 2
#define buttonKey2 4 //connect buttonKey 2 to arduino pin 4
//pins are defined

// make servo object/element, for first servo motor
Servo servo1;
Servo servo2;
int pos = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600); //9600 baud rate (for printing on screen)
  // Note: There is no pulldown resistor, therefore the input options are:
  // INPUT (high resistance 100 Mega ohms, therefore detects a little current easily)
  // or INPUT_PULLUP (20k ohms resistance, this effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on)
  // therefore when key button is pressed it will read LOW, otherwise it will read HIGH
  pinMode(buttonKey1, INPUT_PULLUP); // configure pin up to high, that buttonKey 1 is connected to, as an input
  pinMode(buttonKey2, INPUT_PULLUP); // configure pin up to high, that buttonKey 2 is connected to, as an input
  //inputs are defined

  // 5 servo motors, defualt position all open (4 finger + thumb), first button pressed, all go down (fingers + thumb) except that first finger
  servo1.attach(9);
  servo2.attach(10);
  // pin 9 of arduino is PWM and I am assigning it to servo1, pin 10 to servo2, pin 11 to servo3, pin 5 to servo4, pin 6 to servo5
}

void loop() {
    // put your main code here, to run repeatedly:
  if (digitalRead(buttonKey1) == HIGH && digitalRead(buttonKey2) == HIGH){
  // if keybutton is pressed (it will read LOW)
    Serial.println("Operation 0");
    // set servo position (20 msec = 20,000 microsecond period)
    servo1.write(0);   // set servo1 position to 0
    servo2.write(0);   // set servo2 position to 0
    delay(1000);
    }


  if (digitalRead(buttonKey1) == HIGH && digitalRead(buttonKey2) == LOW){
  // if keybutton is pressed (it will read LOW)
    Serial.println("Operation 1");
    servo1.write(180);   // set servo1 position to 180
    servo2.write(180);   // set servo2 position to 180
    delay(1000);
    }

  if (digitalRead(buttonKey1) == LOW && digitalRead(buttonKey2) == HIGH){
  // if keybutton is pressed (it will read LOW)
    Serial.println("Operation 2");
      for (pos = 0; pos <= 180; pos += 1) {   // goes from 0 degrees to 180 degrees
      // in steps of 1 degree
      servo1.write(pos);                      // tell servo1 to go to 0 to 180
      servo2.write(pos);                      // tell servo2 to go to 0 to 180
      delay(100);                            // waits 100ms for the servo to reach the position
    }
  }

  if (digitalRead(buttonKey1) == LOW && digitalRead(buttonKey2) == LOW){
  // if keybutton is pressed (it will read LOW)
    Serial.println("Operation 3");
      for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
      servo1.write(pos);              // tell servo1 to go from 180  to 0
      servo2.write(pos);              // tell servo2 to go to 180 to 0
      delay(100);
    }
  }
    delay(100); // so button is not displayed/pressed more than once
}
```

26 Arduino Uno on COM5

Figure 26 - *Servo Motor + Switch Code*

- Used two switches (named buttonKey1 & buttonKey2 in *Figure 26*, later renamed to Switch1 & Switch2 in *Figure 28*) as a "binary counter", for example **Operation 0, 1, 2**, and **3** (making a **total of 4 Operations**, similar to the 4-digit keypad).
- Outputted PWM signal to two Servo Motors at a time (for testing), later changed to all 5 Servos (see *Figure 32*)

Switch Schematic:

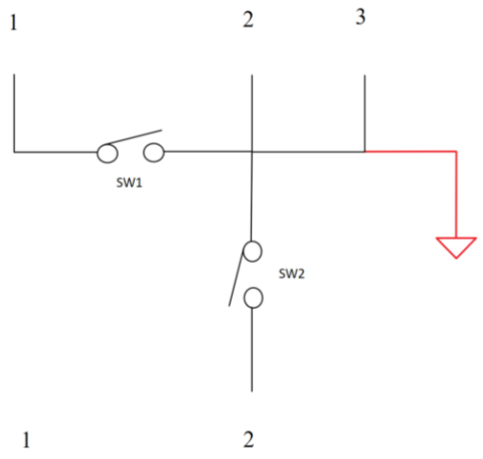


Figure 27 - Two out of the Four Switch Module's Schematic (Two-Bit Binary Counter)

- Used two switches (SW1 & SW2) as a Two-Bit (0, 1, 2, & 3) counter
- buttonKey1 = SW1 (LEFT) = pin 1 from top part of SW1 goes to pin 2 of Arduino
- buttonKey2 = SW2 (RIGHT) = pin 2 from bottom part of SW2 goes to pin 4 of Arduino
  (these two switches are later called Switch1 & Switch2 instead of buttonKey1 & 2 in the next code (see *Figure 28*))
- There are four switches in total on this module, but sw3 & sw4 = not used
- pin 2 connects to pin 3 on top part of switch, and then pin 3 is connected to ground / common
- this is a 4-switch, 14 pin total [7 on each side (top and bottom)] module

Switch Binary Counter Table:

SW1 (IP1) = Left Switch (pin 2)	SW2 (IP2) = Right Switch (pin 4)
0	0
0	1
1	0
1	1

Table 1 - Switch Binary Counter

- IP1 & IP2 from switch module connected to pins 2 and 4 of Arduino UNO
- Two-Bit (0, **1**, 2, & **3** Operations) Counter

- Two ways to go from Closed Fist (Default = 180°) to Open Fist **via Switch**: Activating SW2 = **Operation1** = Moves servos to Open Fist (0°) position immediately, and Activating both SW2 (Right) & SW1 (Left) = **Operation3** = Servo goes from Closed (Default = 180°) to Open Fist (0°) (but closes slowly, automatically increments by 1°, then repeats)

Added Code for PB Counter Control:

I also provided PB control via Pins 7-8 on the Arduino. They are connected to the LEFT & RIGHT push buttons. Note that the pins on the Arduino can be configured as either inputs or outputs. Note that pinMode() defaults as an input, so they don't need to be explicitly declared as inputs. Therefore, since I configured pins 7 & 8 as digital inputs with internal pullup resistors, the Arduino will use internal pull up resistors (20K pullup resistors built into the Arduino's Atmega IC). The other side of the push button must be connected to ground / common in order to receive a LOW signal from the PB.

Two_push_buttons_Servo_Motor_Default_180_Deg_CW | Arduino 1.8.12 (Windows Store 1.8.33.0)

File Edit Sketch Tools Help

Two_push_buttons_Servo_Motor_Default_180_Deg_CW

```
  /*
 Controlling a servo with two Push Buttons (incremental) or Two Switches (operaitonal) via Arduino
  when Right  push button is pressed, the servo start moving to the right until it reaches 180( or zero) degrees
  when Left  push button is pressed, the servo start moving to the left until it reaches 180( or zero) degrees
  when switch is set to binary value (0,1,2,3) it will perform 4 operations
  */

#include <Servo.h> //include servo library
// Note:
//Use PWM pins for servos, PWM pins = 3, 5, 6, 9, 10, & 11, I have 5 servo motors: 4 fingers + 1 thumb
Servo servo1;  // initalize/create servo object to control the servo
Servo servo2;
int angle = 180;    // initial angle  for servo
int angleStep = 5; // when PB is pressed, this is the step it will increment by to the left/right

//Use non-PWM pins as Switch and PB pins, non-PWM pins = 2, 4, 7, 8, 12, 13
#define Switch1 2  // pin 2 is connected to left switch
#define Switch2 4  // pin 4 is connected to right switch

#define LEFT 7   // pin 7 is connected to left push button
#define RIGHT 8  // pin 8 is connected to right push button

// cw aka right string = loose
// ccw aka left string = tight

void setup() {
  Serial.begin(9600); //  setup serial baud rate
  servo1.attach(9);  // attach the servo pwm to pin 9
  servo2.attach(10);  // attach the servo pwm to pin 9
  pinMode(2,INPUT_PULLUP); // sets up internal pull up resistor to 5V, so we just ahve to put the signal to GND to complete signal
  pinMode(4,INPUT_PULLUP); // INPUT_PULLUP (20k ohms resistance, this effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on)
  pinMode(7,INPUT_PULLUP);
  pinMode(8,INPUT_PULLUP);
  Serial.println("Pav's Servo Control via Switch / Push Button ");
}

void loop() { // run continously
  while(digitalRead(LEFT) == LOW){ // when push button (pin 7) is not pressed, it is considered high, aka not true, and the while loop will not be run
  if (angle >= 0 && angle <= 180) {  //sets limits
  angle = angle + angleStep; //increment desired angle up by 5 deg as long as (while) PB is being pressed, 0 -> 180
  if(angle > 180){
     angle = 180;
    }
  else {
      servo1.write(angle); // move the servo to desired angle
      servo2.write(angle);
      Serial.print("Moved to: ");
      Serial.print(angle);   // prints the angle
      Serial.println(" degree");
       } // end of else loop
    } // end of if loop
  delay(100); // waits 100msec for the servo to get there
  }// end of while loop for LEFT PB

  while(digitalRead(RIGHT) == LOW){ //when push button (pin 8) is not pressed, it is considered high, aka not true, and while loop will not be run
  if (angle > 0 && angle <= 180) {  //sets limits
  angle = angle - angleStep; //increment desired angle up by 5 deg as long as (while) PB is being pressed, 180 -> 0
  if(angle < 0){
     angle = 0;
    }
  else {
      servo1.write(angle); // move the servo to desired angle
      servo2.write(angle);
      Serial.print("Moved to: ");
      Serial.print(angle);   // prints the angle
      Serial.println(" degree");
       } // end of else loop
    } // end of if loop
  delay(100); // waits 100msec for the servo to get there
  }// end of while loop for RIGHT PB
```

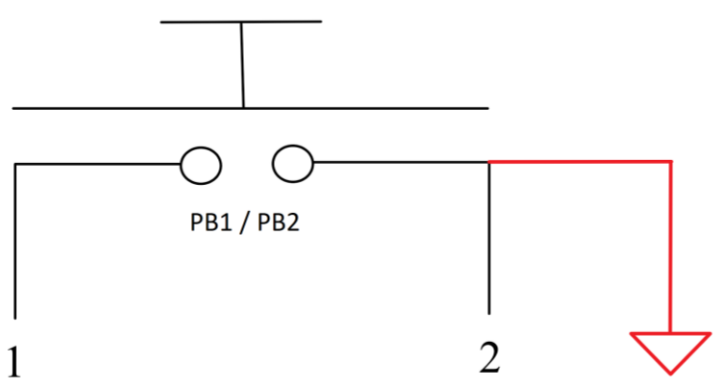
```
while(digitalRead(LEFT) == HIGH && digitalRead(RIGHT) == HIGH){ // When either PB is not active, use switches for control operation
if (digitalRead(Switch1) == HIGH && digitalRead(Switch2) == HIGH){ // neither switches pressed = 00
// if switch is pressed (it will read LOW), if nt pressed it will read high
  Serial.println("Operation 0"); //nothing pressed
  // set servo angleition (20 msec = 20,000 microsecond period)
  servo1.write(180);   // set servo1 angle to 180
  servo2.write(180);   // set servo2 angle to 180
  delay(1000);
  }

if (digitalRead(Switch1) == HIGH && digitalRead(Switch2) == LOW){ // sw2 pressed = 01
// if keybutton is pressed (it will read LOW)
  Serial.println("Operation 1"); // sw2 pressed
  servo1.write(0);   // set servo1 angle to 0
  servo2.write(0);   // set servo2 anglen to 0
  delay(1000);
  }

if (digitalRead(Switch1) == LOW && digitalRead(Switch2) == HIGH){ // sw1 pressed = 10
// if keybutton is pressed (it will read LOW)
  Serial.println("Operation 2"); // sw1 pressed
    for (angle = 0; angle <= 180; angle += 1) {   // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    servo1.write(angle);                      // tell servo1 to go to 0 to 180
    servo2.write(angle);                      // tell servo2 to go to 0 to 180
    delay(100);                             // waits 100ms for the servo to reach the angleition
  }
}

if (digitalRead(Switch1) == LOW && digitalRead(Switch2) == LOW){ // sw1 & sw2 pressed = 11
// if keybutton is pressed (it will read LOW)
  Serial.println("Operation 3");  // sw1 & sw2 pressed
    for (angle = 180; angle >= 0; angle -= 1) { // goes from 180 degrees to 0 degrees
    servo1.write(angle);              // tell servo1 to go from 180  to 0
    servo2.write(angle);              // tell servo2 to go to 180 to 0
    delay(100);
  }
}
  delay(100); // so button is not displayed/pressed more than once
} // end of switch while loop

} // end of continous void loop
```

```
Done uploading.
Sketch uses 4434 bytes (13%) of program storage space. Maximum is 32256 bytes.
Global variables use 346 bytes (16%) of dynamic memory, leaving 1702 bytes for local variables. Maximum is 2048 bytes.
73                                                                                                  Arduino Uno on COM3
```

(Control Servo with Push buttons using Arduino, 2019)

<u>Figure 28</u> – Servo Motor + Switch + LEFT & RIGHT PB's Code

- The motors start at 180° because I changed my design to a closed fist orientation.
- If LEFT or RIGHT PB is LOW, then that means they are pressed or active. If they are HIGH, then they are not pressed or inactive. Therefore, the first highlights shows operation if LEFT PB is held and second highlight shows operation if RIGHT PB is held. Third highlight (while loop for PBs) is to do nothing (remain at 180°). Switch operation is same as in pervious figures.
- After that highlight is the code for the switches (**Operations 0 – 3, total of 4**), **Operation 0 =** immediately to 0°, **Operation 1 =** immediately to 180°, **Operation 2 =** counter to 0°, and **Operation 3 =** counter to 180°



<u>Figure 29</u> - PB Schematic

RIGHT PB being held shown via Arduino Serial Monitor:

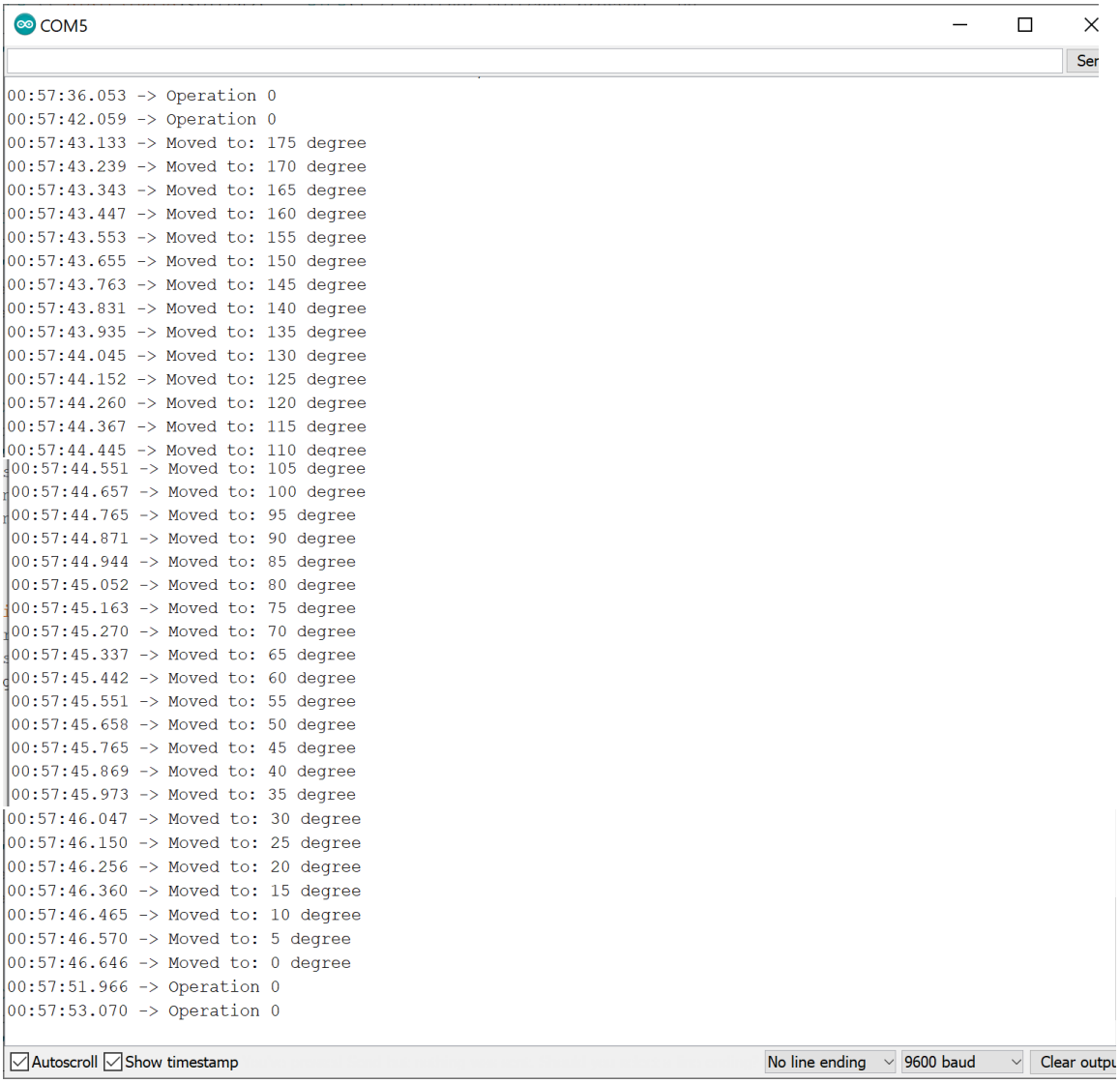
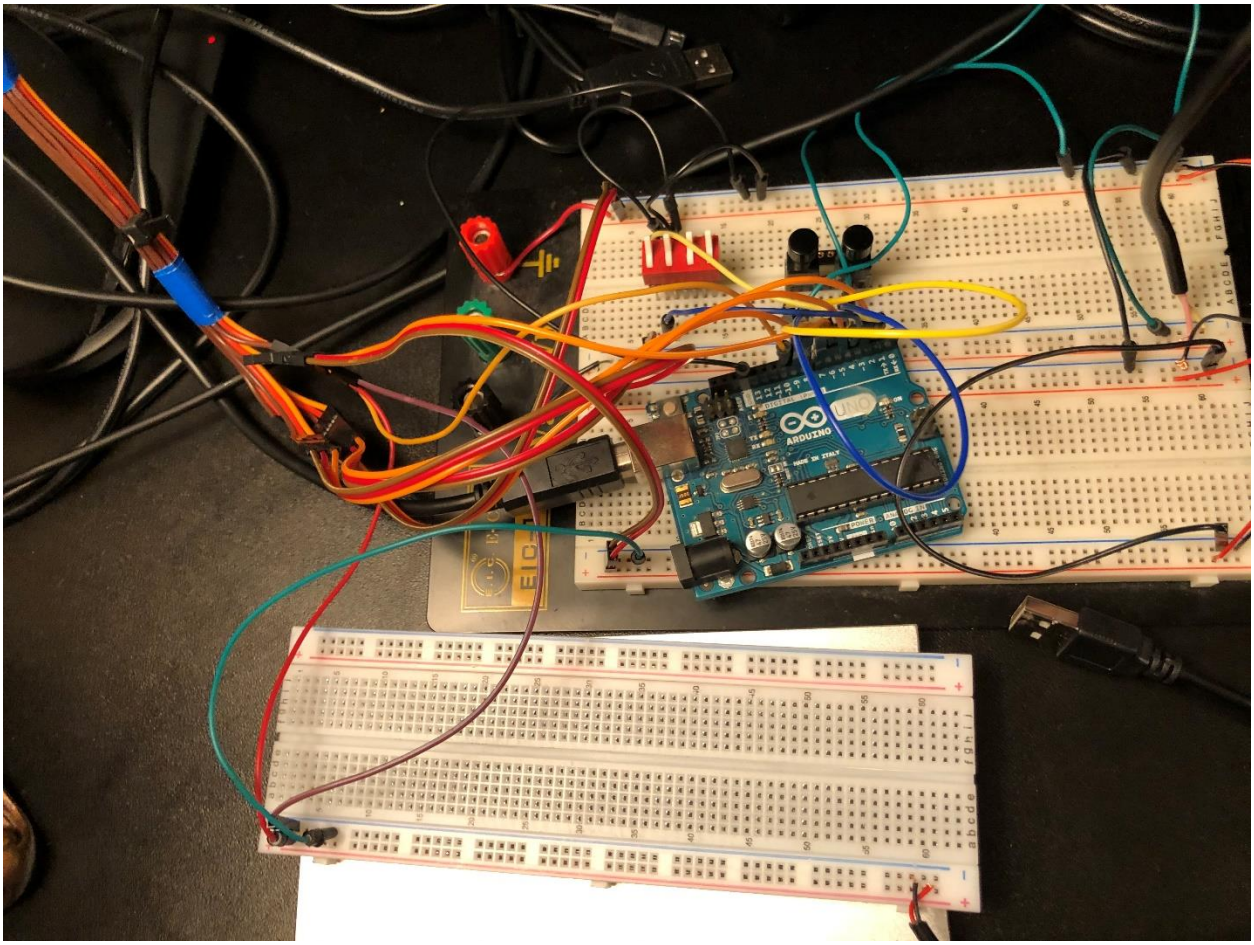


Figure 30 - Serial Plotter of RIGHT PB being held

- Therefore, Servo motor is moving from 180° to 0° (in 5° increments, aka 'angleStep')
- If instead LEFT PB was being held, Servo motor stepping from 0° to 180° (Moved to: 5°, 10°, 15°, …, 180°)

Final Setup: Servo Motors + Switch + PBs:



<u>Figure 31</u> – Final Servo Motors + Switch + PBs Setup

- Servo Motors get PWM signals via orange jumpers connected to Arduino UNO's PWM output pins 3, 5, 6, 9, & 10
- Servo motors power via USB cables to portable charger and snipped end to breadboard (red + black jumpers)

Final Code: 5 Servo Motors + Switch + PBs:

Two_push_buttons_5_Servo_Motors_Default_180_Deg_CW | Arduino 1.8.12 (Windows Store 1.8.33.0)

File Edit Sketch Tools Help

Two_push_buttons_5_Servo_Motors_Default_180_Deg_CW

```
  /*
 Controlling a servo with two Push Buttons (incremental) or Two Switches (operaitonal) via Arduino
  when Right  push button is pressed, the servo start moving to the right until it reaches 180( or zero) degrees
  when Left  push button is pressed, the servo start moving to the left until it reaches 180( or zero) degrees
  when switch is set to binary value (0,1,2,3) it will perform 4 operations
  */

#include <Servo.h> //include servo library
// Note:
//Use PWM pins for servos, PWM pins = 3, 5, 6, 9, 10, & 11, I have 5 servo motors: 4 fingers + 1 thumb, so I will need 5 PWM signals
Servo servo1;  // initalize/create servo object to control each servo
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;
int angle = 180;    // initial angle  for servo
int angleStep = 5; // when PB is pressed, this is the step it will increment by to the left/right

//Use non-PWM pins as Switch and PB pins, non-PWM pins = 2, 4, 7, 8, 12, 13
#define Switch1 2  // pin 2 is connected to left switch
#define Switch2 4  // pin 4 is connected to right switch

#define LEFT 7   // pin 7 is connected to left push button
#define RIGHT 8  // pin 8 is connected to right push button

// cw aka right string = loose
// ccw aka left string = tight

void setup() {
  Serial.begin(9600); //  setup serial baud rate
  servo1.attach(3);  // attach the servo to pwm pin 3 = pinky
  servo2.attach(5);  // attach the servo to pwm pin 5 = ring
  servo3.attach(6);  // attach the servo to pwm pin 6 = middle
  servo4.attach(9);  // attach the servo to pwm pin 9 = index
  servo5.attach(10);  // attach the servo to pwm pin 10 = thumb
  pinMode(2,INPUT_PULLUP); // sets up internal pull up resistor to 5V, so we just ahve to put the signal to GND to complete signal
  pinMode(4,INPUT_PULLUP); // INPUT_PULLUP (20k ohms resistance, this effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on)
  pinMode(7,INPUT_PULLUP);
  pinMode(8,INPUT_PULLUP);
  Serial.println("Pav's Servo Control via Switch / Push Button ");
}

void loop() { // run continously
  while(digitalRead(LEFT) == LOW){ // when push button (pin 7) is not pressed, it is considered high, aka not true, and the while loop will not be run
  if (angle >= 0 && angle <= 180) {  //sets limits
  angle = angle + angleStep; //increment desired angle up by 5 deg as long as (while) PB is being pressed, 0 -> 180
  if(angle > 180){
     angle = 180;
    }
  else {
      servo1.write(angle); // move the servo to desired angle
      servo2.write(angle);
      servo3.write(angle);
      servo4.write(angle);
      servo5.write(angle);
      Serial.print("Moved to: ");
      Serial.print(angle);   // prints the angle
      Serial.println(" degree");
       } // end of else loop
    } // end of if loop
  delay(100); // waits 100msec for the servo to get there
  }// end of while loop for LEFT PB

  while(digitalRead(RIGHT) == LOW){ //when push button (pin 8) is not pressed, it is considered high, aka not true, and while loop will not be run
  if (angle > 0 && angle <= 180) {  //sets limits
  angle = angle - angleStep; //increment desired angle up by 5 deg as long as (while) PB is being pressed, 180 -> 0
  if(angle < 0){
     angle = 0;
    }
  else {
      servo1.write(angle); // move the servo to desired angle
      servo2.write(angle);
      servo3.write(angle);
      servo4.write(angle);
      servo5.write(angle);
      Serial.print("Moved to: ");
      Serial.print(angle);   // prints the angle
      Serial.println(" degree");
       } // end of else loop
    } // end of if loop
  delay(100); // waits 100msec for the servo to get there
  }// end of while loop for RIGHT PB

  while(digitalRead(LEFT) == HIGH && digitalRead(RIGHT) == HIGH){ // When either PB is not active, use switches for control operation
  if (digitalRead(Switch1) == HIGH && digitalRead(Switch2) == HIGH){ // neither switches pressed = 00
  // if switch is pressed (it will read LOW), if nt pressed it will read high
    Serial.println("Operation 0"); //nothing pressed
```

```
  // set servo angleition (20 msec = 20,000 microsecond period)
  servo1.write(180);   // set servo1 angle to 180
  servo2.write(180);   // set servo2 angle to 180
  servo3.write(180);   // set servo3 angle to 180
  servo4.write(180);   // set servo4 angle to 180
  servo5.write(180);   // set servo5 angle to 180
  delay(1000);
  }

if (digitalRead(Switch1) == HIGH && digitalRead(Switch2) == LOW){ // sw2 pressed = 01
// if keybutton is pressed (it will read LOW)
  Serial.println("Operation 1"); // sw2 pressed
  servo1.write(0);   // set servo1 angle to 0
  servo2.write(0);   // set servo2 angle to 0
  servo3.write(0);   // set servo3 angle to 0
  servo4.write(0);   // set servo4 angle to 0
  servo5.write(0);   // set servo5 angle to 0
  delay(1000);
  }

if (digitalRead(Switch1) == LOW && digitalRead(Switch2) == HIGH){ // sw1 pressed = 10
// if keybutton is pressed (it will read LOW)
  Serial.println("Operation 2"); // sw1 pressed
    for (angle = 0; angle <= 180; angle += 1) {   // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    servo1.write(angle);                    // tell servo1 to go to 0 to 180
    servo2.write(angle);                    // tell servo2 to go to 0 to 180
    servo3.write(angle);                    // tell servo3 to go to 0 to 180
    servo4.write(angle);                    // tell servo4 to go to 0 to 180
    servo5.write(angle);                    // tell servo5 to go to 0 to 180
    delay(100);                             // waits 100ms for the servo to reach the angleition
  }
}

if (digitalRead(Switch1) == LOW && digitalRead(Switch2) == LOW){ // sw1 & sw2 pressed = 11
// if keybutton is pressed (it will read LOW)
  Serial.println("Operation 3");  // sw1 & sw2 pressed
    for (angle = 180; angle >= 0; angle -= 1) { // goes from 180 degrees to 0 degrees
    servo1.write(angle);              // tell servo1 to go from 180  to 0
    servo2.write(angle);              // tell servo2 to go to 180 to 0
    servo3.write(angle);              // tell servo3 to go to 180 to 0
    servo4.write(angle);              // tell servo4 to go to 180 to 0
    servo5.write(angle);              // tell servo5 to go to 180 to 0
    delay(100);
  }
}
  delay(100); // so button is not displayed/pressed more than once
} // end of switch while loop

} // end of continous void loop

Sketch uses 4492 bytes (13%) of program storage space. Maximum is 32256 bytes.
Global variables use 355 bytes (17%) of dynamic memory, leaving 1693 bytes for local variables. Maximum is 2048 bytes.

136                                                                    Arduino Uno on COM5
```

<u>Figure 32</u> - Final Code for 5 Servo Motors + Switch + PBs

- All 5 Servos being used, connected to PWM Pins 3, 5, 6, 9, and 10
- **180°** = **Closed Fist** (**default**) & **0°** = **Open Fist**
- Remember LEFT PB = Increments up by 5°, up to 180° (To Closed Fist)
- <u>Remember **RIGHT PB = Increments down by 5°**, down **to 0°** (**To Open Fist**)</u>
- Added 3 more Servos (total of 5) to PB code above
- **Operation 0 (default)** = Nothing pressed (neither PB nor Switch) = **remains Closed Fist** (180°)
- **<u>Operation 1 = SW2 (Right) pressed = Moves servos to Open Fist (0°) position immediately</u>**
- Operation 2 = SW1 (Left) pressed = Servo goes from Open (0°) to Closed (180°) slowly
  (Closes fist slowly, automatically increments by 1°, then repeats)
- **Operation 3 = SW2 (Right) & SW1 (Left) pressed** = Servo **goes from Closed (180°) to Open Fist (0°)** (Closes the fist **slowly**, automatically **increments by 1°**, then repeats)

EXPERIMENTAL WORK – WRIST (STEPPER MOTOR)

Wrist – 28BYJ-48 5V Stepper Motor + ULN2003 Driver Board

This is a 5 Line (Blue, Pink, Yellow, Orange, & Red), “4 Phase”, 5V DC Stepper Motor. I chose a stepper motor for the wrist than a servo motor, because a stepper motor can spin continuously. This feature pairs well with a joystick. Also, stepper motors are much quicker than servo motors. The only downside to a stepper motor is accuracy / position, which wasn’t a huge concern for the wrist.

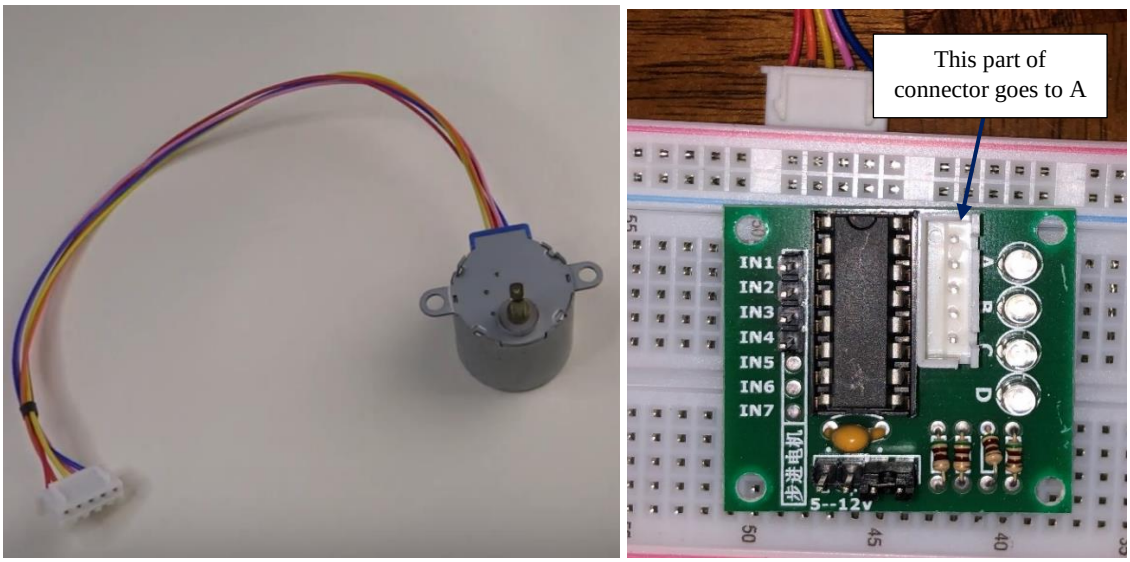


Figure 33 - 5V Stepper Motor 28BYJ-48 and ULN2003A Driver Board

- The 16-pin IC is called ULN2003A and it’s first 4 pins are available inputs for the board
- These 4 pins will connect to the Arduino UNO’s digital pins
- Note 5V Jumpers are open for driver board (12V jumpers are closed off with a cap)
- A, B, C, and D Bits = Coil 4 (Blue), Coil 3 (Pink), Coil 2 (Yellow), and Coil 1 (Orange)
- The Red wire is for the Common signal (see *Figure 34 & 35*) for all 4 Coils, aka a 5V signal
- And for full step operation we need to turn one bit HIGH (5V) at a time (see *Figure 36*)

Side Note: The ULN2003 Driver Board actually supports 5V up to 12V as a power source, however, from the 28BYJ-48 Stepper Motor datasheet (see Appendix) it says the stepper motor requires only 5V. Therefore, I split a USB cable connected to a portable charger’s 5V output and put it into a breadboard rail for the 5V DC connection.

28BYJ-48 5V Stepper Motor Calculations:

- The 28BYJ-48 has 2048 steps / revolution.
- This motor's minimum time between each step is about 2 milliseconds (can't go any quicker/lower).
- Its Max Speed is 500 steps / second

Therefore, for one full rotation / revolution, we need to let it run for about 4 seconds.

Cycle = Speed * Time

$$500\frac{steps}{second} * 4\ seconds = 2000 \text{ steps for one revolution}$$

ULN2003A Schematic

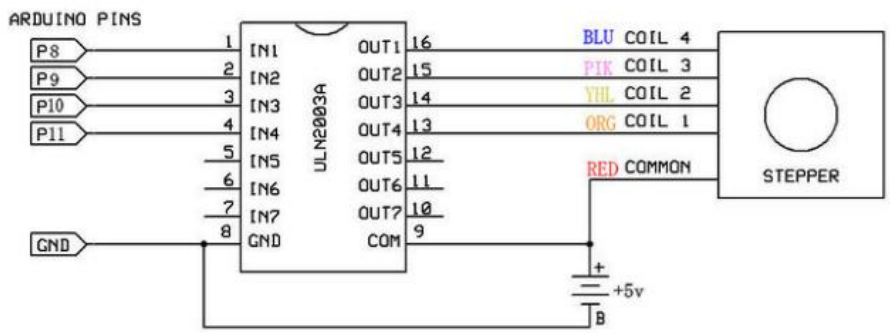


Figure 34 - ULN2003A's Schematic

(Wece, 2018)

- That 5V connection is the same as mentioned above (5V jumpers, instead of 12V, connected to 5V port on portable charger via USB cable)
- 28BYJ-48 Stepper Motor's Driver is called the ULN2003A (the C shown in schematic above)

The Arduino 5V pin allows about 400mA to be outputted, I used an external power supply instead. The external power supply I chose were 20,000 mAh portable wireless chargers, with two USB output ports (one being 1A and the other 2.1A)

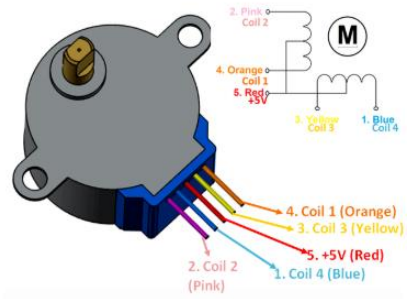


Figure 35 - 28BYJ-48 Stepper Motor Coil Labels

A stepper motor works by moving in steps. Its full steps can be controlled via the 4 available pins on the ULN2003 Driver Board. See below for table of the 4 inputs (IN1 - IN4).

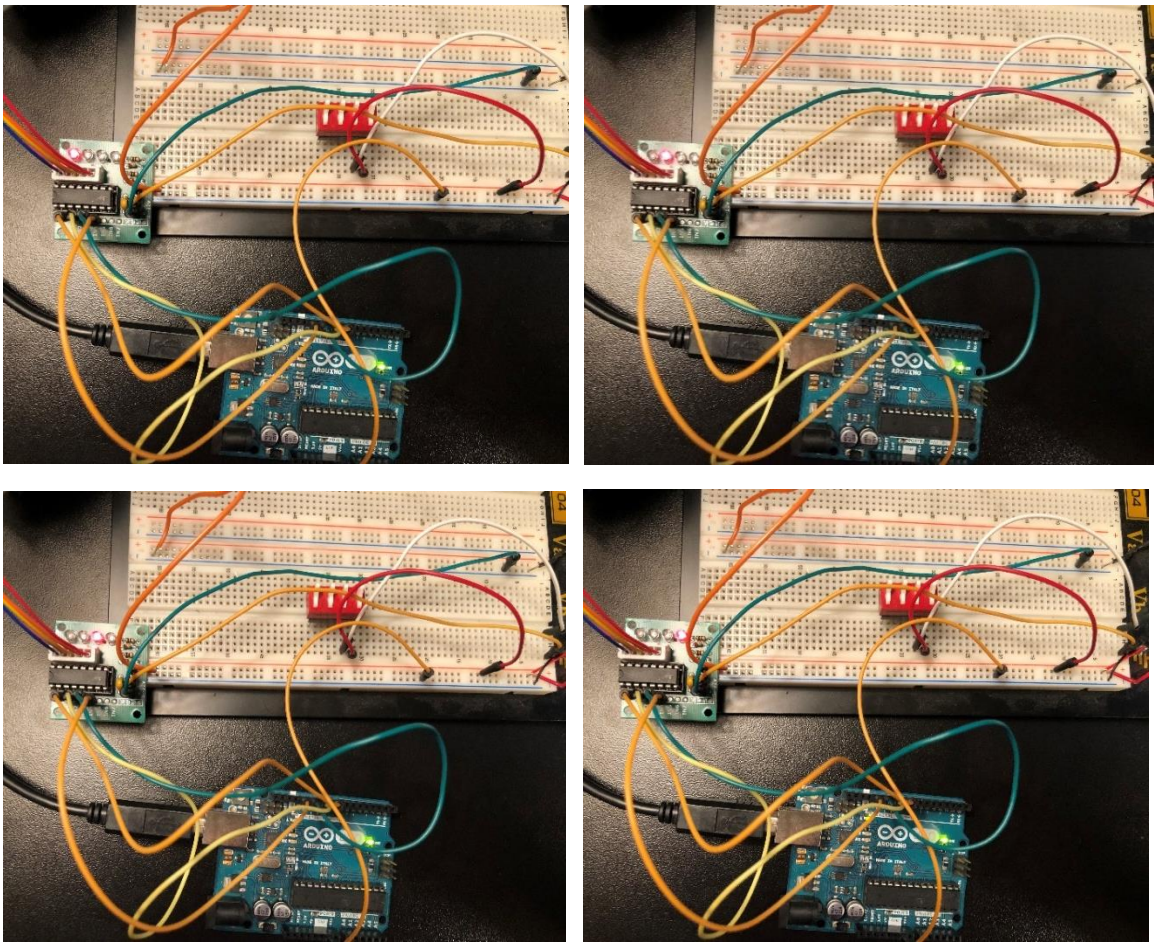
For FULL STEP operation, we must set one of the inputs to HIGH and the rest to LOW, then we must shift that HIGH signal each STEP. After cycle is completed, it is repeated.

STEP	ONE	TWO	THREE	FOUR
IN1	1	0	0	0
IN2	0	1	0	0
IN3	0	0	1	0
IN4	0	0	0	1

(Bartnik, 2018)

Table 2 – 28BYJ-48 Stepper Motor's Wave Driving Table

- To create the first step (STEP ONE), IN1 must receive a HIGH signal and IN2 - IN4 must receive LOW signals.
- **Pins 2, 4, 7, 8**, and **12**, and **13** are **good choices as digital pins** because **3, 5, 6, 9, 10, 11** are the only **PWM capable pins** on the Arduino, so those are better off saved to output to the Servos.
  Therefore, we have to set those pins on the Arduino as outputs (so they can deliver the signal to ULN2003A's IN1 - IN4) via the pinMode() command, and then to send the signal we must use digitalWrite() command.



<u>Figure 36</u> – ULN2003 Driver Board slowed down to show A, B, C, and D Bits / Coils activating one at a time (Red LED)

- In the code we have to shift the HIGH bit by one to activate each coil/phase ("4 Phases" = Blue/Pink/Yellow/Orange Coils, can see that sequential order being followed left to right) and get the stepper motor to spin.
- Arduino needs to output 4 digital signals (one HIGH bit and all other LOW bits, then cycling the HIGH bit) to driver inputs 1 - 4, which then in return will drive each coil of the stepper motor to get rotation.

Note: These pictures are with a delay of 1 second between each step, normally it would be a delay of 2 milliseconds between each step and it would look like all the LEDs are on a time.

28BYJ-48 Stepper Motor Code – Wave Driving

Stepper_Motor4.ino | Arduino 1.8.12 (Windows Store 1.8.33.0)

File Edit Sketch Tools Help

Stepper_Motor4.ino

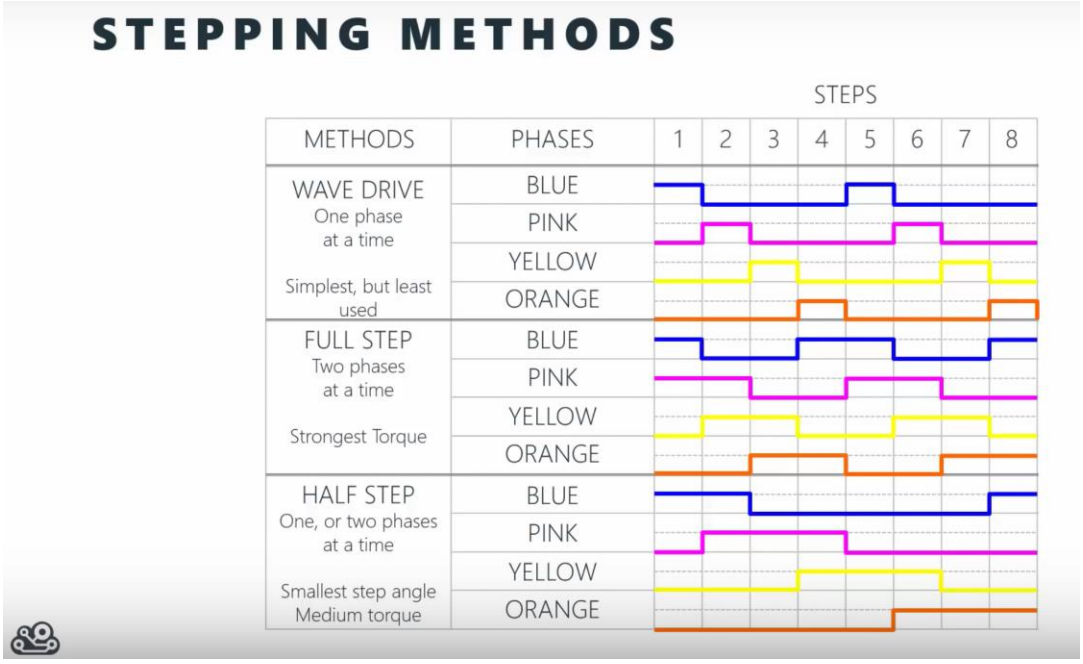
```
// ULN2003A Driver has 4 Digital Inputs that need to be set HIGH and LOW to get FULL STEP operation from the Stepper Motor
// The 28BYJ-48 Stepper Motor has 2048 steps / revolution
// To complete FULL STEPs, the following sequence must be applied to the 4 Stepper Input Pins:
// STEP ONE: IN1 = 1, IN2 = 0, IN3 = 0, IN4 = 0
// STEP TWO: IN1 = 0, IN2 = 1, IN3 = 0, IN4 = 0
// STEP THREE: IN1 = 0, IN2 = 0, IN3 = 1, IN4 = 0
// STEP FOUR: IN1 = 0, IN2 = 0, IN3 = 0, IN4 = 1

// The Arduino will output a HIGH or LOW signal, according to sequence above, to IN1-IN4
// 11.25 Degree Step Angle, 32 steps needed for 360 Degree rotation (32 steps / rev)
// Since rotor shaft has 64:1 ratio with motor shaft, the motor will need 32 steps * 64 = 2048 steps per motor 360 degree rotation
// That means the step above was be divided by 64, 11.25 Degree Step Angle / 64 = 0.18 Degree Step Angle
// Wave Drive = Point at Center of Tooth (Done by shifting 1 High register at a time)
// Full Step = Point inbetween 2 Tooths (Done by shifting 2 High registers at a time)

// ULN2003A Driver has 4 Digital Inputs that need to be set HIGH and LOW to get FULL STEP operation from the Stepper Motor
// The 28BYJ-48 Stepper Motor has 2048 steps / revolution
// To complete FULL STEPs, the following sequence must be applied to the 4 Stepper Input Pins:
// STEP ONE: IN1 = 1, IN2 = 0, IN3 = 0, IN4 = 0
// STEP TWO: IN1 = 0, IN2 = 1, IN3 = 0, IN4 = 0
// STEP THREE: IN1 = 0, IN2 = 0, IN3 = 1, IN4 = 0
// STEP FOUR: IN1 = 0, IN2 = 0, IN3 = 0, IN4 = 1
// The Arduino will output a HIGH or LOW signal, according to sequence above, to IN1-IN4

// Note:
//Use non-PWM pins as keyboard pins, non-PWM pins = 2, 4, 7, 8, 12, 13, and 2, 4, 7, 8 were used in Servo Motor Code, that leaves 12 and 13 open
// Therefore we will use Digital Pins 9, 10, 12, and 13 as the Digital Outputs for the HIGH/LOW signals required for the above sequence
#define STEPPER_PIN_1 9 // digital output pin connected to ULN2003A IN1
#define STEPPER_PIN_2 10 // digital output pin connected to ULN2003A IN2
#define STEPPER_PIN_3 12 // digital output pin connected to ULN2003A IN3
#define STEPPER_PIN_4 13 // digital output pin connected to ULN2003A IN4
int step_number = 0;  // currently step 0 out of 4
bool dir; // define dir variable
//pins numbers are defined

void setup() {
  // Put my setup code here, so it runs once:
  Serial.begin(9600); //9600 baud rate (in case I need printing on screen)
  pinMode(STEPPER_PIN_1, OUTPUT);
  pinMode(STEPPER_PIN_2, OUTPUT);
  pinMode(STEPPER_PIN_3, OUTPUT);
  pinMode(STEPPER_PIN_4, OUTPUT);
  //pins numbers are defined as outputs


  digitalWrite(STEPPER_PIN_1, LOW);
  digitalWrite(STEPPER_PIN_2, LOW);
  digitalWrite(STEPPER_PIN_3, LOW);
  digitalWrite(STEPPER_PIN_4, LOW);
  //pins are initialized to zero
 }

void loop() { // main code that is run repeatidly, sets dir to true for half cycle, and false for other cycle

switch(step_number){ //instead of a bunch of if statmernts, I used a switch case (its similar to if statemnts but a lot cleaner and concise)
  //step_number is the value I'd like to compare
  case 0: //case 0 is when step_number = 0
    digitalWrite(STEPPER_PIN_1, HIGH); // energize blue coil
    digitalWrite(STEPPER_PIN_2, LOW);
    digitalWrite(STEPPER_PIN_3, LOW);
    digitalWrite(STEPPER_PIN_4, LOW);
    break; //exit case 0
  case 1: //case 1 is when step_number = 1
    digitalWrite(STEPPER_PIN_1, LOW);
    digitalWrite(STEPPER_PIN_2, HIGH); // energize pink coil
    digitalWrite(STEPPER_PIN_3, LOW);
    digitalWrite(STEPPER_PIN_4, LOW);
    break; //exit case 1
  case 2: //case 2 is when step_number = 2
    digitalWrite(STEPPER_PIN_1, LOW);
    digitalWrite(STEPPER_PIN_2, LOW);
    digitalWrite(STEPPER_PIN_3, HIGH); // energize yellow coil
    digitalWrite(STEPPER_PIN_4, LOW);
    break; //exit case 2
  case 3: //case 3 is when step_number = 3
    digitalWrite(STEPPER_PIN_1, LOW);
    digitalWrite(STEPPER_PIN_2, LOW);
    digitalWrite(STEPPER_PIN_3, LOW);
    digitalWrite(STEPPER_PIN_4, HIGH); // energize orange coil
    break; //exit case 3
  // Basically CASE 0 -> CASE 3 = STEP ONE -> STEP FOUR (See truth table)
}
  step_number = (++step_number < 4) ? step_number : 0;
  delay(1000); // 1000msec = wait 1 second between each step
  // delay(2) = 2000 microseconds, or 2 milliseconds (to get motor to spin fast) which is about the shortest delay that is usable to be able to point to the next tooth accuratly.
  // Anything lower and the motor starts to freeze and gets stuck.
}
```

82 Arduino Uno on COM5

Figure 37 - *Stepper Motor – Wave Driving Code*

- Note: Table shows **STEP 1 → 4** (see *Table 2*) which in the code (figure above) translates to **Case 0 → 3**
- **Step 1** = Case 0 = **Energize Blue Coil**, **Step 2** = Case 1 = **Energize Pink Coil**, **Step 3** = Case 2 = **Energize Yellow Coil**, and **Step 4** = Case 3 = **Energize Orange Coil**

The table (see *Table 2*) showed one bit activating at a time, this is called Wave Driving. That is also what the code (in *Figure 37*) shows above as well. The figure below shows that same table (Wave Driving) but in a wave format rather than a table.



<u>Figure 38</u> – Stepper Motor – Wave Driving (one phase on at a time and 11.25 Step Angle)

- Wave Driving = One HIGH bit at any given time
- Full Step = Two HIGH bits at any given time
- I tested both Wave Driving and Full Step but neither produced enough torque to move the hand

28BYJ-48 Stepper Motor Code – Full Stepping

Full Stepping

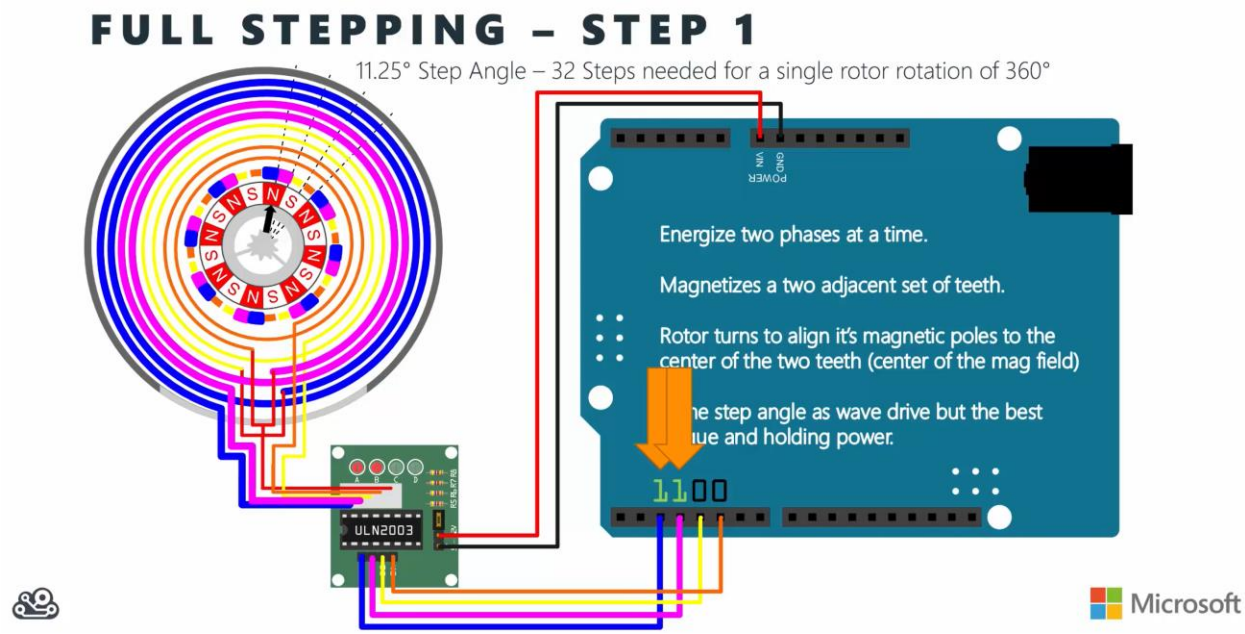


Figure 39 – Stepper Motor – Full Stepping (Two phases on at a time and 11.25 Step Angle)

- Shows that Blue and Pink phases are both active at a time, next would be Pink and Yellow, then Yellow and Orange, and lastly Blue and Orange
- Since two coils are on at a time, magnetic field is twice as strong, which means it'll have more torque
- This, however, was still not enough torque to move the hand

Wrist – Joystick Schematic and Wiring

Wrist was supposed to rotate via the **Joystick** & **Stepper Motor**, see above for Stepper motor code (*Figure 37*) and below for Joystick code (*Figure 42*).

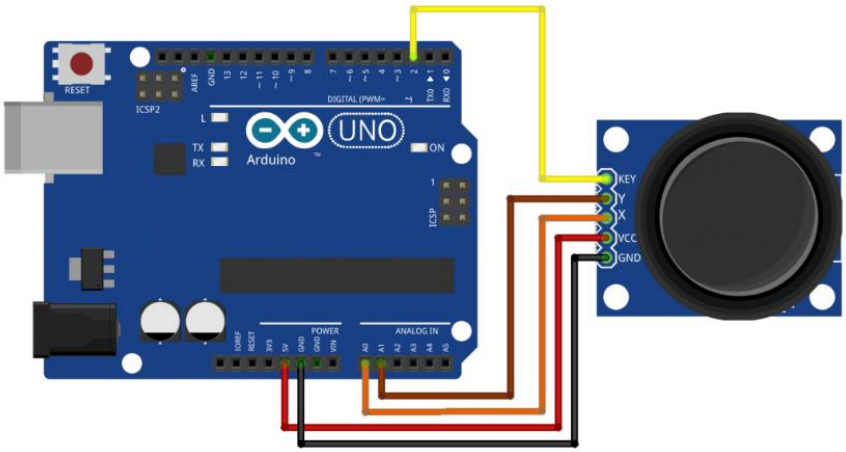


Figure 40 – Arduino UNO and Joystick Wiring

(Brainy Bits, 2015)

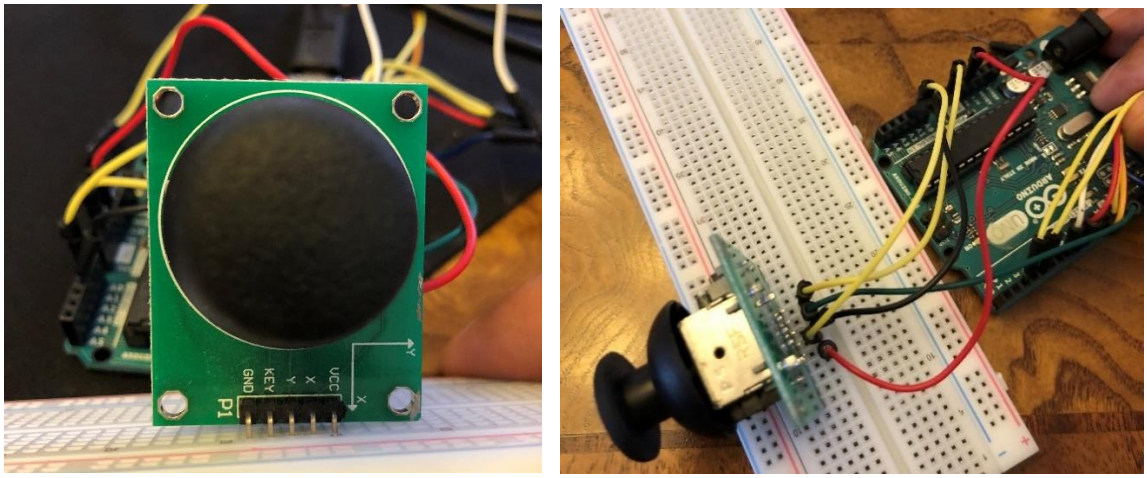


Figure 41 - Arduino UNO and Joystick Actual Wiring

Wrist – Joystick Code on Arduino UNO

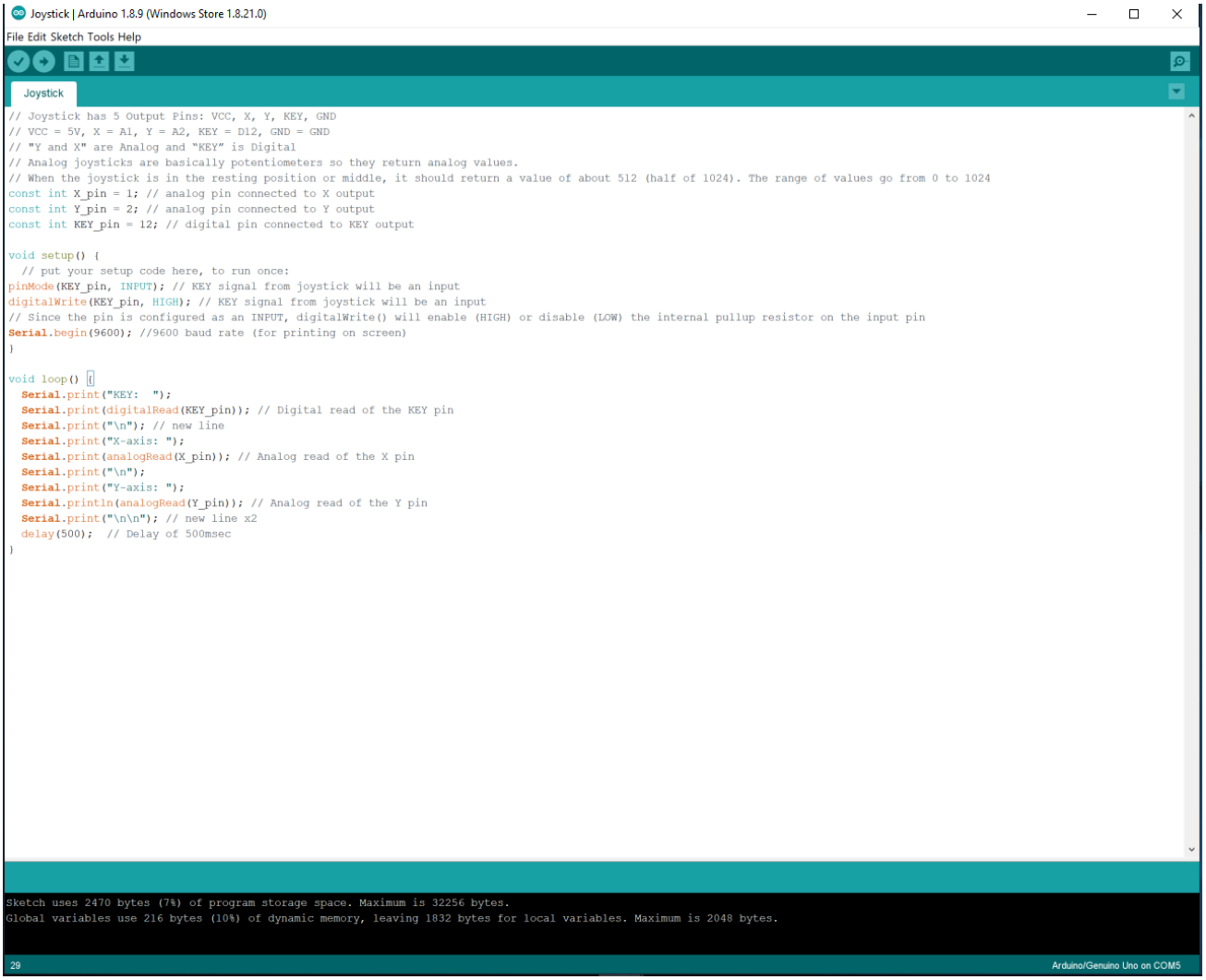


Figure 42 - Arduino UNO read code for joystick

- Got help from Brainy Bits (Brainy Bits, 2015)
- Reads results of Joystick operation (see *Figure 43 – Figure 47)*

Wrist – Joystick Read Results when Joystick is clicked

Clicking the joystick resets values to zero, resting middle values X-axis = 522 & Y-Axis = 513

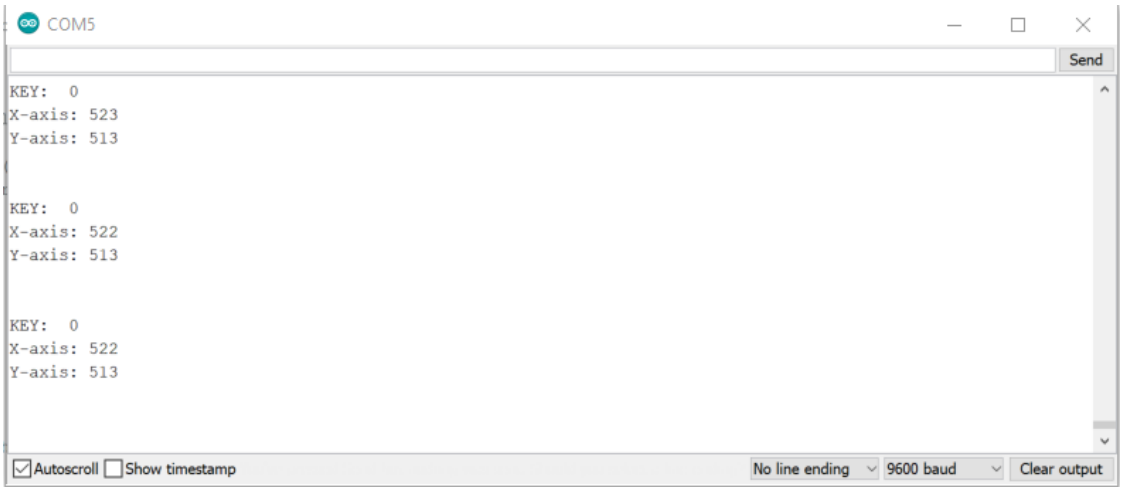


Figure 43 - Arduino UNO code result for when Joystick is clicked

Wrist – Joystick Read Results moving to extremes: All the way UP, DOWN, LEFT, or RIGHT

Moving Joystick All the Way Up: Y-axis = 1024

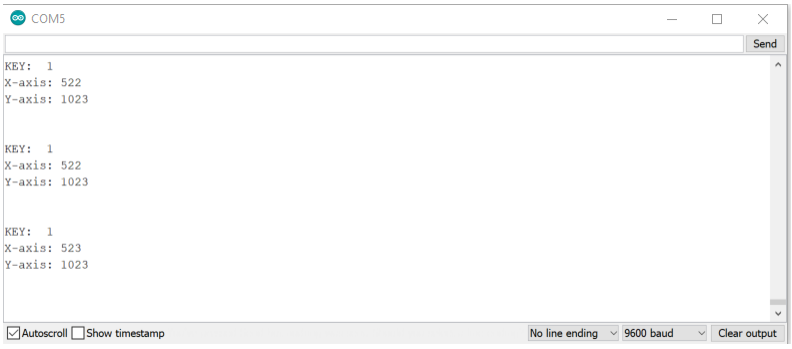


Figure 44 - Arduino UNO code result when Joystick moved All the Way Up

Moving Joystick All the Way Down: Y-axis = 0

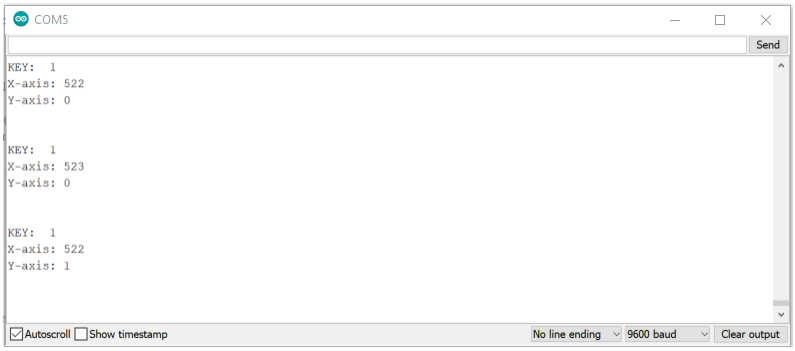


Figure 45 - *Arduino UNO code result when Joystick moved All the Way Down*

Moving Joystick All the Way Left: X-axis = 1024

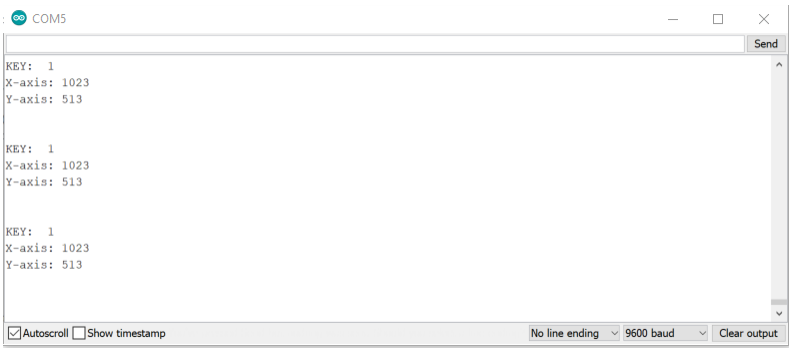


Figure 46 - *Arduino UNO code result when Joystick moved All the Way Left*

Moving Joystick All the Way Right: X-axis = 0

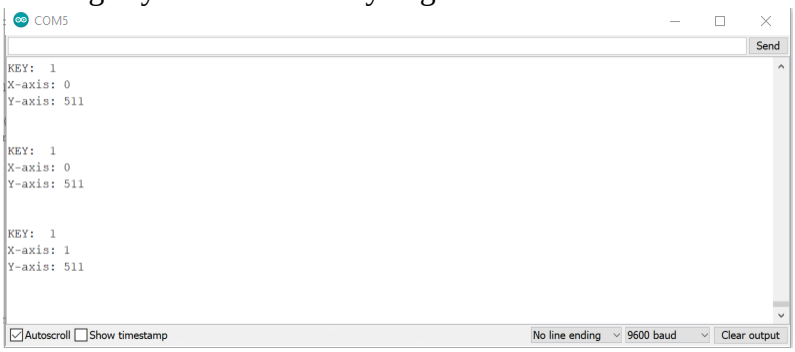


Figure 47 - *Arduino UNO code result when Joystick moved All the Way Right*

EXPERIMENTAL WORK - OBSERVATIONS AND ISSUES:

I planned to use 1 Stepper Motor (for wrist), and not a 6th Servo motor, because I wanted continuous rotation (like a robot) and to control that rotation with a joystick. In hindsight I would have attempted to mod the Servo Motor for 360-degree rotation or just use it as 180 deg rotation (with the 6th PWM signal available on the Arduino). The Stepper motor on the other hand, doesn't use PWM, and uses 4 digital signals in order to moves in steps. Its full steps can be controlled via the 4 available pins on the ULN2003 Driver Board (see *Figure 34* & *Table 2*).

Servo Motors are a lot more durable and have a lot more torque, I guess that's why they were more expensive (rightly so) shown in the cost breakdown. Next time, I would probably use a higher quality stepper motor or just use a 6th servo motor directly mounted (with no attachment) onto the palm/wrist connector (show in *Figure 9* and not use the modified one in *Figure 13*). Stepper motors draw less current and has a higher range of motion, and that is why I was initially hoping for them to work out. I did not expect **spacing**, **weight**, and **procedural** steps as much as I should have.

The **spacing** would have helped place/mount the motors in specific spots. They could have been concealed entirely in the forearm. Also, the fishing line's spacing could have been separated in pairs for each of the five motors that make up the hand:

The **weight** was an issue for the Stepper Motor (Wrist), it did not have enough torque to spin the wrist. This was confirmed via connecting the Arduino directly to Stepper Motor's ULN2003 Driver Board and outputting to a Stepper Motor without the 3D Printed Components – it spun around fine.

This then led to a subsequent issue of not being able to use a joystick (see *Figures 40 & 41*) to control the Stepper Motor. The joystick was intended to be used as a potentiometer and telling the Stepper Motor to spin faster or slower in the clockwise or counterclockwise direction (see *Figures 43-47* for the joystick reads). The Servo motors on the other hand can only rotate about 180 degrees – not continuously.

CONCLUSIONS

7 things learned while completing this project:

- Servo Motors are accurate in position but generally only are able to rotate about 180 degrees.
- Stepper Motors are fairly accurate in position and can rotate continuously, however they do not produce as much torque, and therefore can't support moving as much weight.
- How to code DC Motors (Servo/Stepper), Joysticks, and Keypads to function with an Arduino.
- How to use internal pull-up resistors with Arduino.
- Fishing Line can withstand a fair bit of tension.
- The difference between .SDL and .STL files.
- That Production Planning is crucial.

PROPOSALS FOR FURTHER WORK / RECOMMENDATIONS

This project went well but it could have gone a lot better. Next time I would probably use the InMoov projects design and get all the required tools before starting. Even if I skipped using the gears, I feel having the motors mounted directly onto each finger (like the InMoov project) allows for better tension and control. My project the fishing lines ran to the forearms. Therefore, I believe that distance and since they weren't properly spaced and mounted, led to a lack of tension.

Next time for the fingers I would ensure constant tension on the fishing line for each motor one step at a time, rather than assembling the entire hand first and placing all the motors motors/fishing lines later. Lastly, for the wrist I would ensure the weight of the entire 3d-printed hand can be supported by the Stepper Motor and then purchase that one accordingly. Or just take a different design approach: different type of palm connector, gears, or simply just using a Servo motor glued onto the palm.

My recommendation is to focus on production planning as much as you can in order to execute a project successfully because it is hard to go back and correct for a change in plan. A great way to achieve this is through BCIT Imaging's 4-C's in order to achieve the 5th C, which are Concept, Consultation, Cost Considerations, and Completion Date, in effort for Client Contentment (BCIT Imaging, 2020).

REFERENCES

28BYJ-48 - 5V Stepper Motor. (n.d.). Retrieved from Components 101: https://components101.com/motors/28byj-48-stepper-motor

Arduino. (2020). *Language Reference*. Retrieved from Arduino Reference: https://www.arduino.cc/reference/en/#page-title

Arduino Uno Tutorial [Pinout]. (n.d.). Retrieved from DIYI0T: https://diyi0t.com/arduino-uno-tutorial/

Bartnik, N. (2018). *28BYJ-48 stepper motor and ULN2003 Arduino*. Retrieved from YouTube: https://www.youtube.com/watch?v=avrdDZD7qEQ&t=50s

BCIT. (2020). *Media Works*. Retrieved from BCIT: https://www.bcit.ca/library/mediaworks.shtml

BCIT Imaging. (2020). Retrieved from BCIT: https://www.bcit.ca/imaging/planning.shtml

Brainy Bits. (2015). Retrieved from How to connect and use Analog Joystick with Arduino: https://www.brainy-bits.com/arduino-joystick-tutorial/

Control a 'LOT' of Servo Motors using a Joystick, Arduino and PCA9685 PWM Module. (2016). Retrieved from Brainy Bits: https://www.brainy-bits.com/control-multiple-servos-with-joystick/

Control Servo with Push buttons using Arduino. (2019). Retrieved from Robojax: https://robojax.com/learn/arduino/?vid=robojax-servo-push-button

Google Images. (2020). Retrieved from Google: https://www.google.com/search?q=carpal+bones+a)+first+superficial+layer+b)+Second+Layer+c)+d)&tbm=isch&ved=2ahUKEwil_bKUitvoAhVVIn0KHbS1BhsQ2-cCegQIABAA&oq=carpal+bones+a)+first+superficial+layer+b)+Second+Layer+c)+d)&gs_lcp=CgNpbWcQA1CMQFiPsAFg_7IBaABwAH

Kiatronics. (n.d.). *28BYJ-48 – 5V Stepper Motor*. Retrieved from RoboCraft: http://robocraft.ru/files/datasheet/28BYJ-48.pdf

Langevin, G. (2020). *Build yours*. Retrieved from InMoov: http://inmoov.fr/build-yours/

Langevin, G. (2020). *InMoov body parts library*. Retrieved from InMoov: http://inmoov.fr/inmoov-stl-parts-viewer/?bodyparts=%2A

learn, S. (2017). *How to wire and code 28BYJ-48 Stepper Motors with an Arduino*. Retrieved from YouTube: https://www.youtube.com/watch?v=HdqCtAJxhrs

MG996R Tower-Pro. (n.d.). Retrieved from Electronicos Caldas: https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf

Solidworks Models. (2017). Retrieved from Google Drive: https://drive.google.com/drive/folders/1e-wlQ0iiDdNyWz2fzkwzrQ9xNT1m0rEW

Stateham, B. (2015). *28BYJ-48 Stepper Motor and ULN2003 Driver Intro*. Retrieved from YouTube: https://www.youtube.com/watch?v=B86nqDRskVU&t=759s

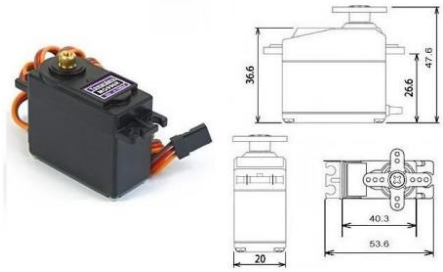
Wece, P. (2018). *Using Arduino*. Retrieved from Arduino: https://forum.arduino.cc/index.php?topic=523604.0

Note: All other pictures, taken from my iPhone.

APPENDIX

TowerPro MG996R Servo Motor Datasheet

MG996R High Torque
Metal Gear Dual Ball Bearing Servo



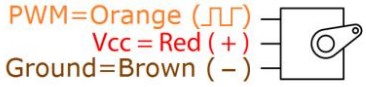
This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

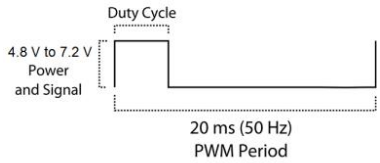
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)

- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA – 900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 °C – 55 °C





(MG996R Tower-Pro, n.d.)

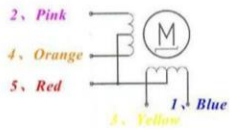
28BYJ-48 Stepper Motor Datasheet

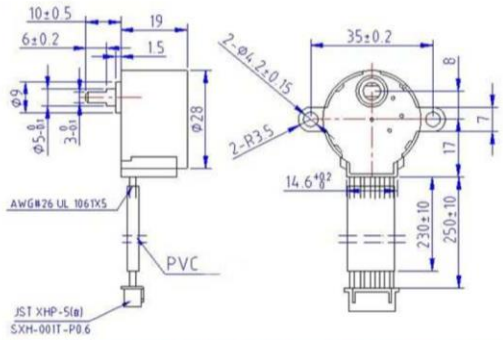
Kiatronics
electronic design and manufacture

28BYJ-48 – 5V Stepper Motor

The 28BYJ-48 is a small stepper motor suitable for a large range of applications.

Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25°C)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V

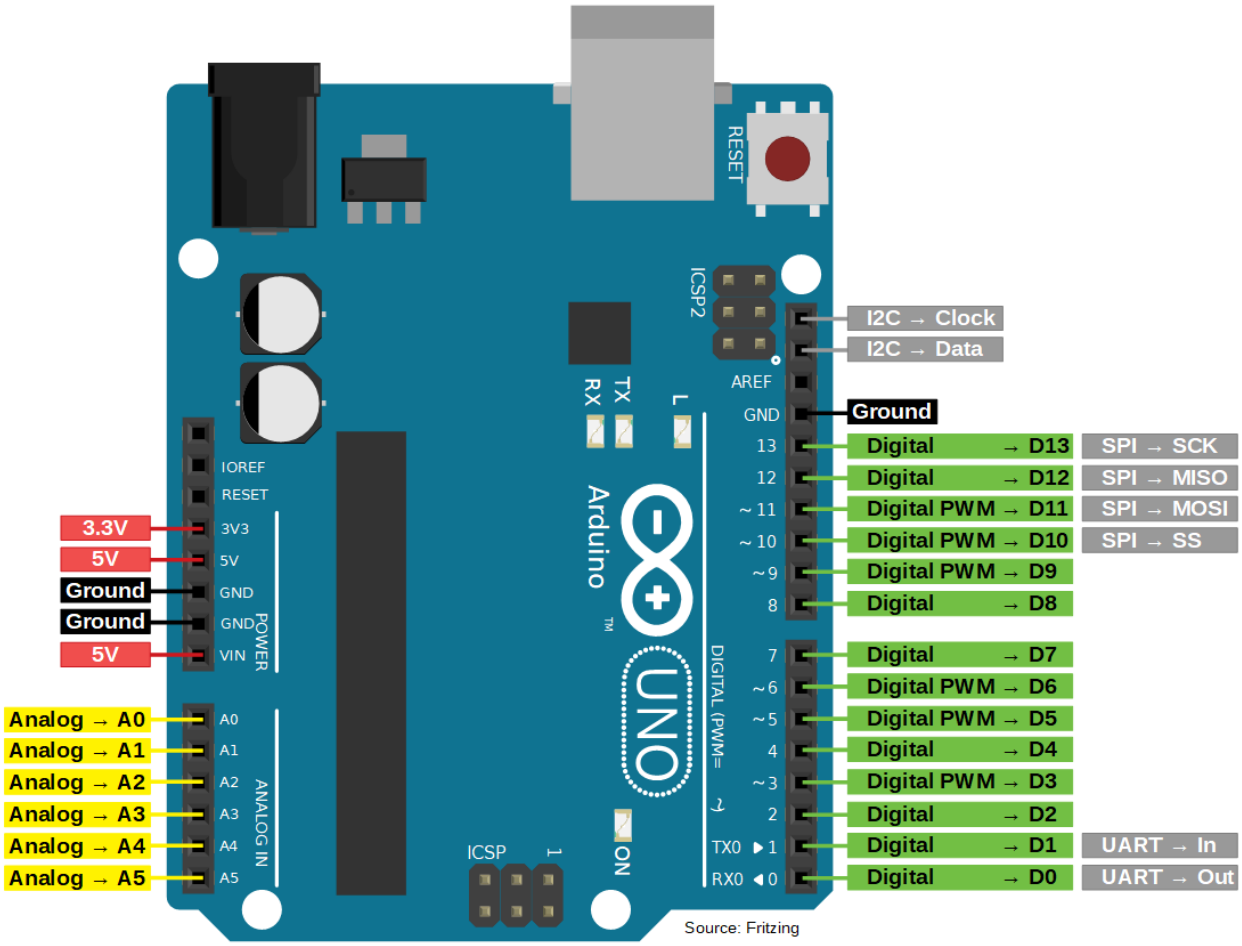




P.O. Box 8231 Cherrywood Tauranga New Zealand Phone: +64 7 578 7739 Fax: ++64 7 578 7749 E-mail: enquiry@kiatronics.com
Website: www.kiatronics.com Copyright © Welton Holdings Ltd - Specifications subject to change without further notice.

(Kiatronics, n.d.)

Arduino UNO Pin Layout



(Arduino Uno Tutorial [Pinout], n.d.)

Notes / Appendix

- HIGH = 5V, LOW = 0V, and there is no negative voltages (e.g. 5V = +5V).
- Arduino Uno Pinout (see Appendix pg.50)
- Servo & Stepper Motor Datasheets (see Appendix pg.50)