Design of a Glove Controller for Human / Robot Interface

By

Gwyer Quest Sinclair

An Honors Capstone
Submitted in partial fulfillment of the requirements
For the Honors Diploma
to

The Honors College

of

The University of Alabama in Huntsville

November 13th, 2018

Honors Capstone Director: Dr. John Piccirillo

____
Student Date

____
Director Date

____
Department Chair Date

____
Honors College Dean Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.
In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

____
Student Name (printed)

____
Student Signature

____
Date

Table of Contents

TERMINOLOGY

Algorithm	– *A technique a programmer could explain, but doesn't want to*
DOF	– *Degrees of Freedom*
I2C	– *Inter-Integrated Circuit Communication Protocol*
IMU	– *Inertial Measurement Unit*
PCB	– *Printed Circuit Board*
PLA	– *Polylactic Acid*
PWM	– *Pulse Width Modulation*
STEM	– *Science, Technology, Engineering and Mathematics Fields*
STL	– *Stereolithography (file format)*

Dedication:

This work is dedicated to the memory of Steve Sinclair, who taught me to imagine fantastic things.

Abstract

This paper discussed a prototype a human interface device, a glove which utilizes various sensors to measure data, process it, and relay commands to robotic hardware. The glove controller will provide an intuitive robotic control to the end user.

Background

There are a number of existing techniques that are used to control arm-like industrial robots. Other robot types (such as mobile robots) may be remote controlled or controlled by internal logic. This project, however, focuses on creating a new control method for industrial robots. The three traditional methods are Walkthrough Training, Pendant control, and Programming.

Walkthrough training features a human 'teacher' physically manipulating the robot's appendages through a set of actions, while the robot records those actions. Later, the actions can be replayed to mimic the task with accuracy. This method is very useful for worker without a technical skillset, as one only needs to physically manipulate the robot. However, not all robots can be manipulated this way as some are too heavy or cumbersome to reliably move, and sometimes there is a safety factor. The robot can be quickly retrained, but it limited by the precision of the user. Users who do not have the strength or dexterity to adjust the robot are not capable of using this control method.

Figure 1: Industrial Robotic Arm

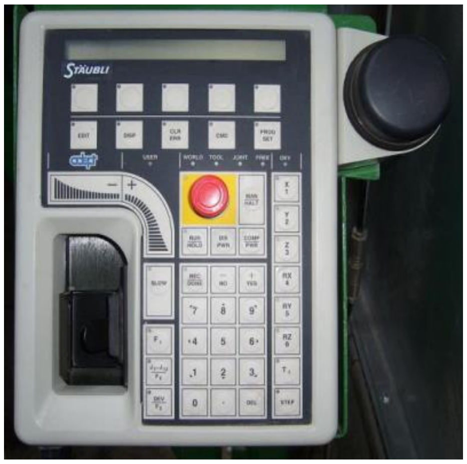


Figure 2: Pendant

Pendant control is the use of a portable, specialized computer to control a robot. A pendant connects to the robot and a technician uses the device to command incremental movement or actions, while the system records the task. Unlike the walkthrough method, pendant training requires a technician to be trained with the individual system, as each robot will use a unique control system. The pendant training method has the advantage of precision over the walkthrough method, but is much slower.

Programming is the use of computer control and simulation to plot a task for a robot. It is similar to the pendant method in that it programs a set of actions directly into the memory, but is often done off-site and without direct access to the robot. Both computerized methods are slower than walkthrough training, and have the advantage of precision control, though direct programming is often the best at creating efficient tasks and coordinating multiple tasks / robot interactions. A disadvantage of programming a robot offsite is the lack of an ability to check your work in real-time. In addition, this method is substantially slower than the others.

This project seeks to extend the main benefit of walkthrough training (that it is accessible to people with non-technical skillsets), and remove the physical limitations. By mapping the movement of a robot arm to the hand's motion, my product would allow those with limited physical mobility or strength to operate robots. In addition, it would increase the safety of workers by keeping them away from the product lines when robots have to be retrained. This project is research into more intuitive and inclusive robotics for all.

Hardware Design

An important factor in product design is ergonomics, and consideration must be made that the product is comfortable and easy to use. I designed the glove with this in mind, keeping it light and efficient by designing robust parts to be manufactured with 3D printing. The parts have been designed to fit my hand, but small adjustments can be made to easily change to another's measurements. I neglected to design a one-size-fits-all item, as the working base for my hand was all that was required to prototype the product and test its efficacy. These parts were designed in AutoCAD Inventor Professional 2018 ™, a software which AutoCAD has made free to students and educators. The images provided are used with attribution to the AutoCAD Software Suite and ViewSTL.com.

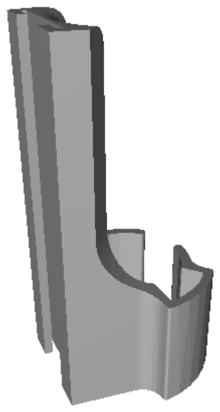


Figure 3: Finger Mount Ver. 5

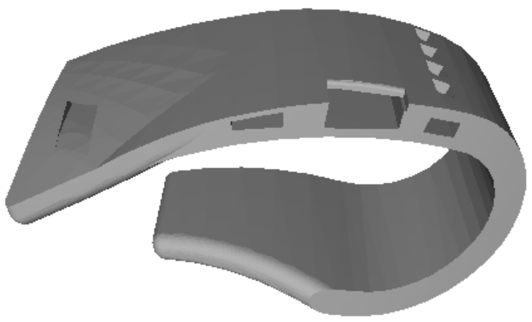


Figure 4: Hand Wrap Ver. 5

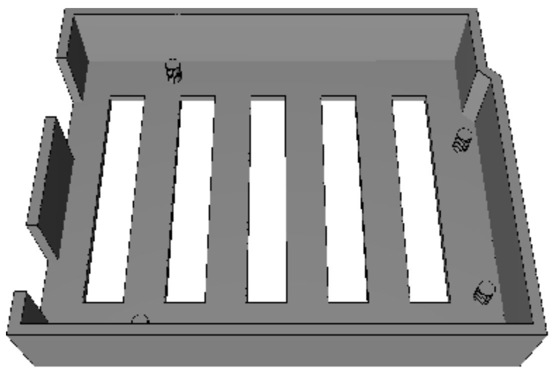


Figure 5: Microcontroller Box Ver. 1

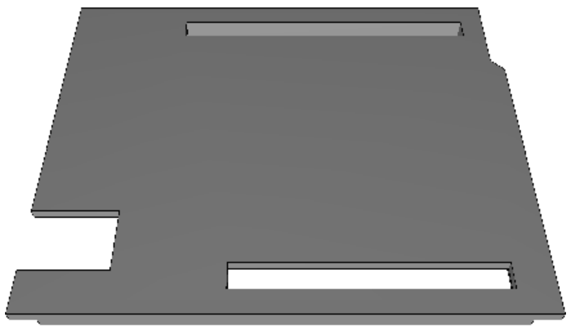


Figure 6: Microcontroller Box Cap Ver. 1

Each piece went through a design process consisting of an initial part and fitting, followed by successive incremental changes and re-printing. The final pieces are shown in Figures 3-6**,** with version numbers provided. Each of these parts was printed from PLA, a commonly used 3D printing thermoplastic that is biodegradable, light, and strong. PLA is also very cheap, and the total cost for all of the prototyped parts comes to just under $5.

Electrical Design

After designing the printed 'shell' of the gloves the next step was to prototype the electrical components. This was a multi-step process involving the selection, testing, and assembly of a sensor package, microcontroller, and connective components. The weight, power consumption, and complexity of the design were all factors in these choices, but even more important was consideration of the intended features of the design.

First I chose sensors which would allow me to gather a range of useful data on the user's motions. These parts are all listed in further specificity in Appendix 1: Parts List and Costs. The general use, design, and parameters of each sensor are included below:

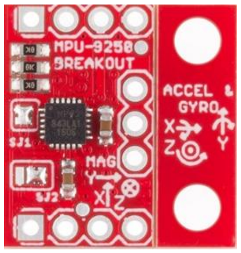


Figure 7: MPU9250

- IMU: The IMU I used was the MPU9250 breakout from Sparkfun electronics. This is a 9 DOF IMU, meaning that it measures acceleration, angular rate, and magnetic field each in three axes. The IMU was placed on the back of the hand, and used to measure the pose angle and side-to-side motion of the hand. This was achieved by use of trigonometry and a high-pass rejection filter, respectively, which is discussed further detail in "Robot Integration and Control", on page 11.

Figure 8: Flex Sensor

- Flex Sensor: A flex sensor is a variable resistor. As the sensor is flexed, bands of metal on its surface get farther apart, increasing its resistance. I mounted the flex sensor across the index finger, effectively measuring the amount of bend in the finger. This was to be mapped to the forward motion of the arm or the closing motion of the claw, depending on individual user preference.

Figure 9: Soft Potentiometer

- Soft Potentiometer: A soft potentiometer is another version of variable resistor, featuring two metal strips that are not in contact. When a force (such as a finger) is placed on a point along its length, the strips are forced together, completing the circuit. The resistance to this current is proportional to the amount of metal the current has to pass through, meaning that touching further along the length increases the resistance. In this way, the soft potentiometer determines where along its length it was touched. I mounted this sensor to the outside of the index finger, where it could be operated the thumb, and mapped it to the open/close position of the robot manipulator.

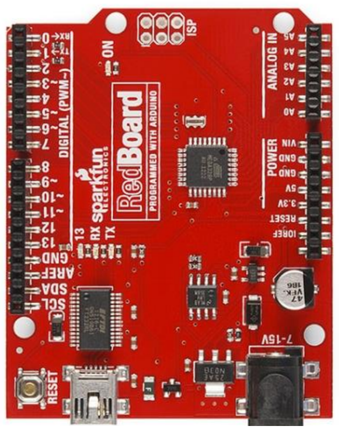


Figure 10: Arduino RedBoard

- Buttons: Buttons are points where the circuit is open, and the user may apply a force to connect it. This allows the button to sense an ‘on’ or ‘off’ state, and an action may be mapped to a button press. I constructed buttons out of copper tape on the back of the hand. Buttons can eventually be used to record or replay actions, or may be mapped to another function.

- Microcontroller: Arduino RedBoard (Figure 10), a microprocessor with 13 Digital I/O Pins (6 PWM Enabled), 6 Analog IN Pins, 3.7 and 5.5 V Power Circuits, and I2C Connection Capability

After selecting my parts and theoretically mapping their measurements to robot actions, I began to prototype this control with the Arduino Microcontroller and software IDE (integrated development environment). This is discussed further in the following section, Software Design.

The only output was the control of 6 servos, which constitute the joints of the robot arm. A servo is a type of motor which can be commanded to rotate to and hold at a certain angle, using PWM control. Using kinematics, any arm position or movement can be broken down into a discrete set of servo angles over time. This kinematic theory is studied in “Robot Integration and Control” on Pages 12-13.

Finally, after each system was tested individually, I attached each sensor to the hardware and completed the assembly (see Appendix 2: Wiring Diagram). This involved cable management and soldering of multiple pull-up resistors in-between wire nodes. A production-level device of this sort would use Arduino’s internal pull-up resistors instead, or those on a PCB. After final assembly was complete, troubleshooting of the software began.

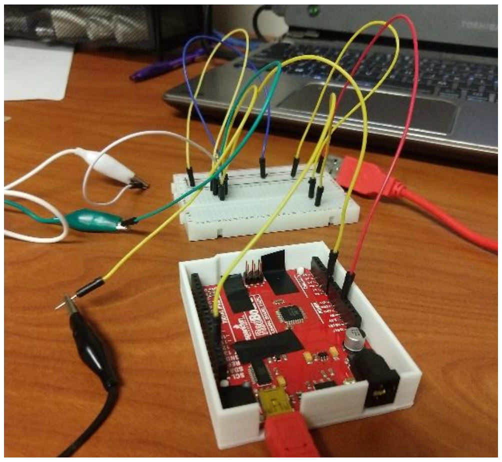


Figure 11: Electrical Prototyping

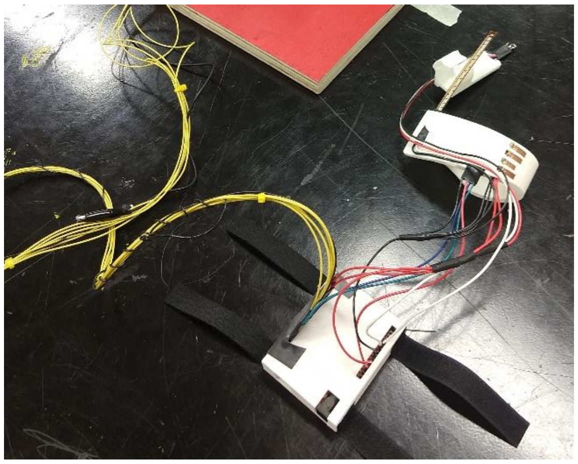


Figure 12: Final Glove Assembly

Software Design

My prototype product is nothing without its 'brain' – the software which recorded and processed inputs, and directed control to the servo motors in the robot arm. I began with individual sections of code which I had formulated to test each individual sensor circuit, and combined them into one program (See Appendix 3: Code). I will briefly discuss the code here in layman's English. The technically-savvy reader should refer to Appendix 3 and view the well-documented[1] code itself.

The code performs a set of initialization tasks on startup, and then a repeating string of tasks in quick succession for as long as it remains on. Upon starting up, the program configures the microcontroller to expect inputs on certain pins (where the sensors are attached) and create outputs on others (the servos). It then carves out space in its memory to store values which correspond to those read from each sensor, the values to be sent to the servos, and numerous variables which will be used for intermediary mathematics. These variables are empty locations in memory for now, later they will be filled and then re-written with their values. After the program is initialized, it enters the 'operational' mode, where the following functions happen in order, and loop back to #1 when complete:

1. Each sensor is polled to find its current value
2. Mathematical operations are performed to determine the output values
3. The output values are sent to each servo, which rotates to match

[1] The code is user-friendly. It is, however, very particular of who its friends are.

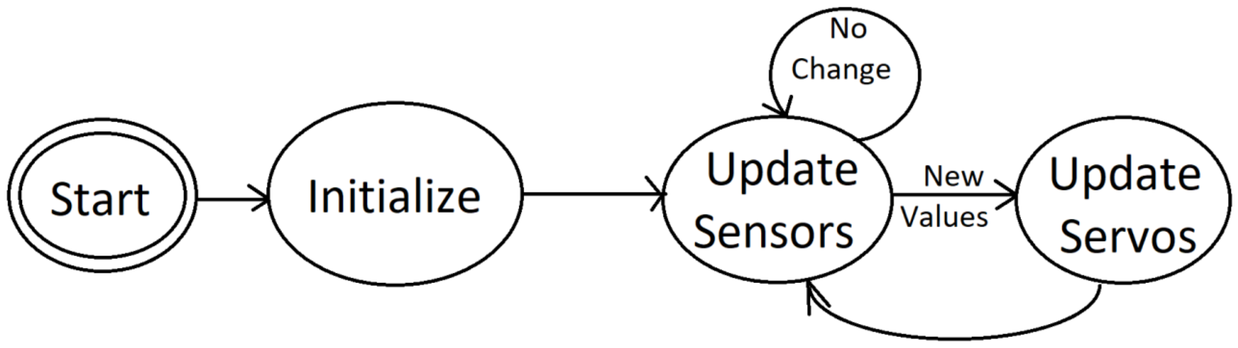


Figure 13: Finite State Machine Diagram

In this way, the code continually updates the position of the arm to the desired position. The program runs thousands of times in one second, meaning that the arm has the illusion of smooth motion.

Robot Integration and Control

The controller measures physical data from the wearer's hand and performs a series of mathematical operations on them to determine what output angles to send to the servos. These operations are discussed in reference to each of the six servos indicated on Figure 14.

<u>Servo 0</u>: "Base"

Servo 0 controls the polar rotation of the arm, and its angle is determined by the left-to-right position of the hand as determined by the IMU. The IMU gyroscope is utilized to measure the angular acceleration about the z-axis (the vertical axis). A simple high-pass check is also utilized, so low values do not register. This is to eliminate the 'noise' inherent in the sensor from interference or a jittery human hand. When the user deliberately moves with an angular acceleration great enough to overcome the filter, a variable is incremented in the direction of motion.

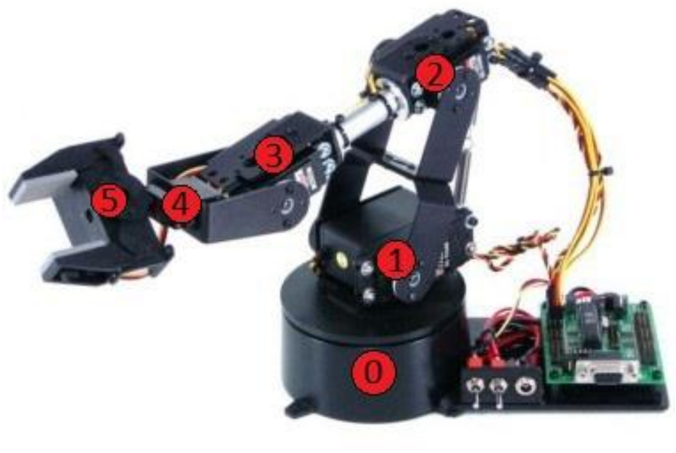


Figure 14: Lynxmotion Robotic Arm (Servos marked)

<u>Servos 1 and 2</u>: “Shoulder” and “Elbow”

Servos 1 and 2 control the forward extension and height of the arm. In this project they are tied together to act in tandem, combining to move the arm out to the desired extension. Due to the limitations of finding absolute position with an IMU without complementary external sensors, these servos are tied to the flex sensor in this implementation. The flex of the finger is mapped to the extension. With servos 1 and 2 both held to the same angle, the “forearm” section of the robot maintains level at all times. This may be altered in future builds to be modified by a ‘height’ parameter, controlled by the gyroscope as in servo 0, which adds the height dimension into the control scheme.

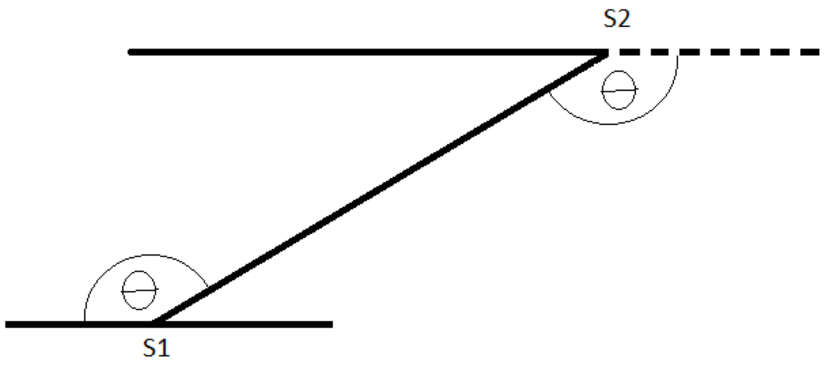


Figure 15: Interaction of Servos 1 and 2

Servos 3 and 4: “Wrist Pitch” and “Wrist Roll”

Servos 3 and 4 control the pose angle of the “wrist”. The angle of each of these servos is determined by a trigonometric manipulation of the acceleration obtained from the IMU’s 3 DOF accelerometer.

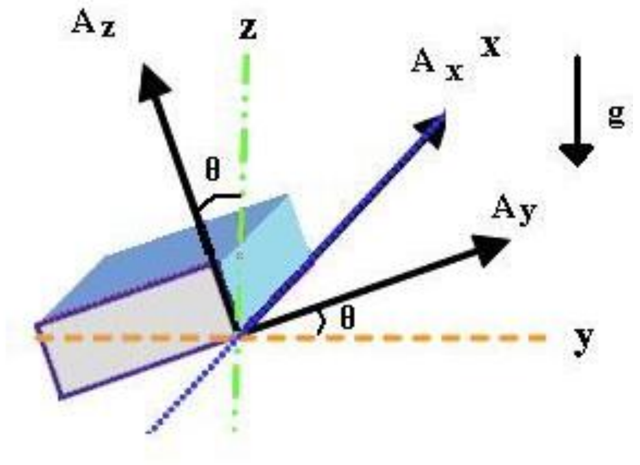


Figure 16: Calculating Angle from Acceleration

$$\theta_R = atan2\left(\frac{Az}{-Ax}\right) \qquad \theta_P = atan2\left(\frac{Az}{-Ay}\right)$$

Servo 5: “Manipulator”

Servo 5 controls the end effector of the robot arm, in this case a claw-like manipulator. This servo is coupled to a linear screw which moves the claw’s fingers from open (0 degrees) to closed (180 degrees). This value is mapped directly from the soft potentiometer. In this way the user can open or close the robot’s fingers by sliding the thumb along the sensor.

Results

The prototyping phase of this project took the majority of my time, and it was not without its challenges. The cycle between hardware part iterations took some time, but I was able to use it to test individual sensor circuit controls. The most challenging part was determining absolute position of the hand from the sensors I had available, and I eventually pivoted to a simpler solution which only limits the glove control in 1 axis – height. This solution allows for smooth control and quick response time, and avoids all errors that would be introduced through the process of determining position through integration of acceleration data. Without utilizing exteroceptive sensors (which acquire information from the environment), the glove measures correct, finely-tunable position in 5 axes and removes all but the most significant human jitter from the arm's motion. Control can be extended to 6 axes in future iterations, as discussed in the previous section.

Conclusion

My controller is a step toward more intuitive control methods for robotics. In fact, this device can be reconfigured to work with any output device, from a drone or mobile robot to a software, VR, or gaming platform. The application of this project to its original design, the control of industrial robots, is particularly of note. Intuitive, non-technical control will become increasingly important in the industry of tomorrow as automation continues to replace human labor. Innovators need to provide solutions for workers with non-technical backgrounds to interface with these new machines, or the demand for STEM trained individuals will quickly outstrip the supply. My glove controller is a step towards more inclusive and intuitive control of the robots of tomorrow.

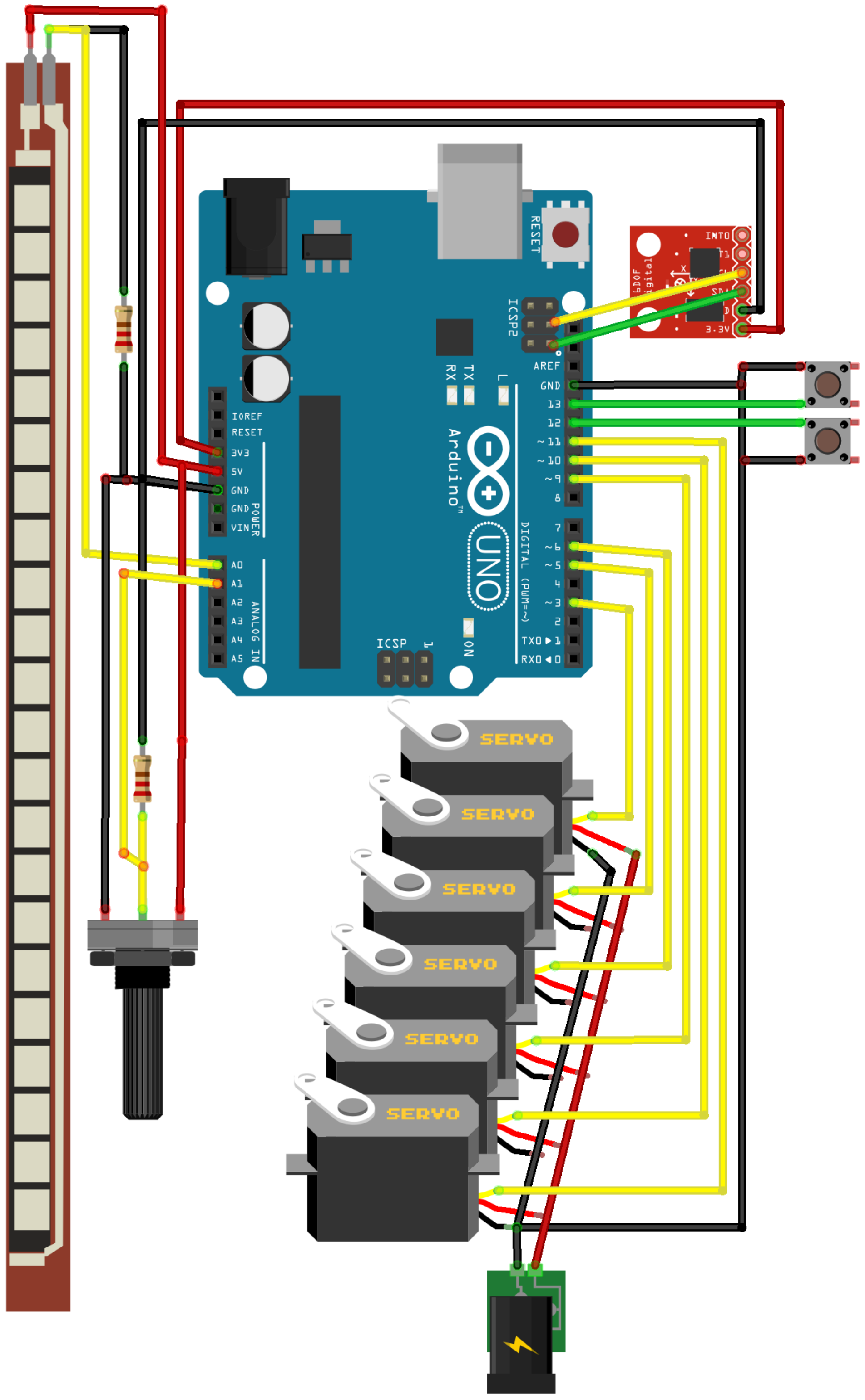
Table of Figures

APPENDIX 1: Parts List

Consumables and parts with fractional cost are omitted (tape, wire, pins, solder, etc)
All costs in US Dollars

Part Name	Type	Source	Quantity / Cost	Notes
Arduino RedBoard	Microcontroller	SparkFun Electronics	1 x 19.95	
Flex Sensor 4.5”	Sensor	SparkFun Electronics	1 x 12.95	
MPU-9250 IMU	Sensor	SparkFun Electronics	1 x 14.95	Breakout Board
PLA Filament	Raw Material	Amazon.com	.25 x 19.99	3D printing material
Soft Potentiometer	Sensor	SparkFun Electronics	1 x 4.95	50 mm length
			~$60	TOTAL COST

APPENDIX 2: Wiring Diagram (made with Fritzing)



APPENDIX 3: Code

```
#include <Wire.h>
#include <TimerOne.h>
#include <Servo.h>

#define    MPU9250_ADDRESS            0x68

#define    GYRO_FULL_SCALE_250_DPS    0x00
#define    GYRO_FULL_SCALE_500_DPS    0x08
#define    GYRO_FULL_SCALE_1000_DPS   0x10
#define    GYRO_FULL_SCALE_2000_DPS   0x18

#define    ACC_FULL_SCALE_2_G        0x00
#define    ACC_FULL_SCALE_4_G        0x08
#define    ACC_FULL_SCALE_8_G        0x10
#define    ACC_FULL_SCALE_16_G        0x18

const int button1Pin = 11;     // fwd copperbutton pin
const int button2Pin = 13;     // back copperbutton 2 pin
const int flexPin = A0;       // Define analog input pin to measure
const int potPin = A1;        // Analog input pin

//int count = 0;

int extPos;                    // how far is the arm extended
int potValue;                    // soft potentiometer
int flexPosition;               // Input value flex sensor
int swivel = 90;                // start val for base rotation (forward)
int claw;                      // claw gripper


int button1State, button2State;                        // variables to hold the pushbutton states

Servo servo0, servo1, servo2, servo3, servo4, servo5;   // declare arm servos


void setup()
{
  Wire.begin();
  Serial.begin(115200);

  // Set accelerometers low pass filter at 5Hz
  I2CwriteByte(MPU9250_ADDRESS,29,0x06);  // thanks to SPARKFUN ELECTRONICS for IMU setup code
  // Set gyroscope low pass filter at 5Hz
  I2CwriteByte(MPU9250_ADDRESS,26,0x06);

  // Configure gyroscope range
  I2CwriteByte(MPU9250_ADDRESS,27,GYRO_FULL_SCALE_1000_DPS);
  // Configure accelerometers range
  I2CwriteByte(MPU9250_ADDRESS,28,ACC_FULL_SCALE_4_G);

  // Set up the pushbutton pins to be an input:
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);

  servo0.attach(3);     // base
  servo1.attach(5);     // shoulder
  servo2.attach(6);     // elbow
  servo3.attach(9);       // wrist pitch
  servo4.attach(11);      // wrist roll
  servo5.attach(10);      // claw

}


// Main loop, read and display data
void loop()
{
  potValue = analogRead(potPin); // Read the voltage from the softpot (0-1023)

  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);

  flexPosition = analogRead(flexPin);

  // Read accelerometer and gyroscope
  uint8_t Buf[14];
  I2Cread(MPU9250_ADDRESS,0x3B,14,Buf);

  // Create 16 bits values from 8 bits data

  // Accelerometer
  int16_t ax=-(Buf[0]<<8 | Buf[1]);
  int16_t ay=-(Buf[2]<<8 | Buf[3]);
  int16_t az=Buf[4]<<8 | Buf[5];

  // Gyroscope
  int16_t gx=-(Buf[8]<<8 | Buf[9]);
  int16_t gy=-(Buf[10]<<8 | Buf[11]);
  int16_t gz=Buf[12]<<8 | Buf[13];

  // calibrate gyroscope values (ROOM TEMPERATURE)
  gx -=48;
  gy += 71;
  gz -= 26;

  // Display values

  // Accelerometer
  /*Serial.print (ax,DEC);
  Serial.print ("\t");
  Serial.print (ay,DEC);
  Serial.print ("\t");
  Serial.print (az,DEC);  */
  /*Serial.print (thetaR);
  Serial.print ("\t");
  Serial.print (thetaP);
  Serial.print ("\t");
  Serial.println();*/

  // Gyroscope
 /* Serial.print (gx,DEC);
  Serial.print ("\t");
  Serial.print (gy,DEC);
  Serial.print ("\t");
  Serial.print (gz,DEC);
  Serial.print ("\t");*/

  extPos = map(flexPosition, 480, 660, 0, 95);  // mapping flex sensor to extension
  extPos = 155-extPos;  // it's a reverse mapping  (low 155 high 60)
  servo1.write(extPos);  // run 1 & 2 at the same angle to stay ~ straight and extend
  servo2.write(extPos);

  float thetaR = degrees(atan2(double(az),double(-ax)));  // find roll angle simple trig
  float thetaP = degrees(atan2(double(az),double(-ay)));  // QED
  servo3.write(thetaP);
  servo4.write(thetaR); // wrist servos

  claw = map(potValue, 0, 1023, 0, 180);  // mapping soft pot to claw
  servo5.write(180-claw);


  if (gz > 1500)         swivel -= 1;   // increment swivel by rotational acceleration
  else if (gz < -1500)   swivel += 1;   // threshold values based on how fast hand has to move

  if (swivel < 25)       swivel = 25;   // limit swivel to physical best spots
  else if (swivel > 155) swivel = 155;
  servo0.write(swivel);

 //delay(100);
 //count++;
  Serial.println("");
}

void I2Cread(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data)  // from sparkfun IMU library
{
  // Set register address
  Wire.beginTransmission(Address);
  Wire.write(Register);
  Wire.endTransmission();

  // Read Nbytes
  Wire.requestFrom(Address, Nbytes);
  uint8_t index=0;
  while (Wire.available())
    Data[index++]=Wire.read();
}


// Write a byte (Data) in device (Address) at register (Register)
void I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data)   // from sparkfun IMU library
{
  // Set register address
  Wire.beginTransmission(Address);
  Wire.write(Register);
  Wire.write(Data);
  Wire.endTransmission();
}
```