Project Final Report

Animatronic Hand Controller

Group-5(Mas-Cube)

Department of EEE, Ahsanullah University of Sciene And Technology,Dhaka,Bangladesh

Team members

Oshin_sabrina@yahoo.com
kazisadman047@gmail.com
mowleemannan@gmail.com
afrozahaque77@gmail.com
ananto.tr@gmail.com

Project Abstract-

For some time, We have been interested in making some sort of robot based on the Arduino platform. This project is the first phase of this longer-term desired effort. Anything is possible with the mighty power of Arduino. It's compact, it's straightforward, and makes embedding electronics into the world-at-large fun and easy.

Animatronics is the use of mechatronics to create machines which seem animate rather than robotic. Animatronic figures are most often powered by pneumatics (compressed air), and, in special instances, hydraulics (pressurized oil), or by electrical means. The figures are precisely customized with the exact dimensions and proportions of living creatures. Motion actuators are often used to imitate "muscle" movements, such as limbs to create realistic motions. Also, the figure is covered with body shells and flexible skins made of hard and soft plastic materials. Then, the figure can be finished by adding details like colors, hair and feathers and other components to make the figure more realistic.

The project idea came to us after watching the movie named "Real Steel". We wanted to make a shadow robot from our curiosity. As the whole body of the robot would have been of much cost, We decided to make a shadow hand instead. Approximating the kinematics of the human hand was our top priority when developing this animatronic hand. Each joint of this hand has a movement range again the same as or very close to that of a human hand, including the thumb and even the flex of the palm for the little finger.

Keywords- **Arduino, Servo motors, Flex sensors, Power IC, Programming of the Arduino.**

I. INTRODUCTION

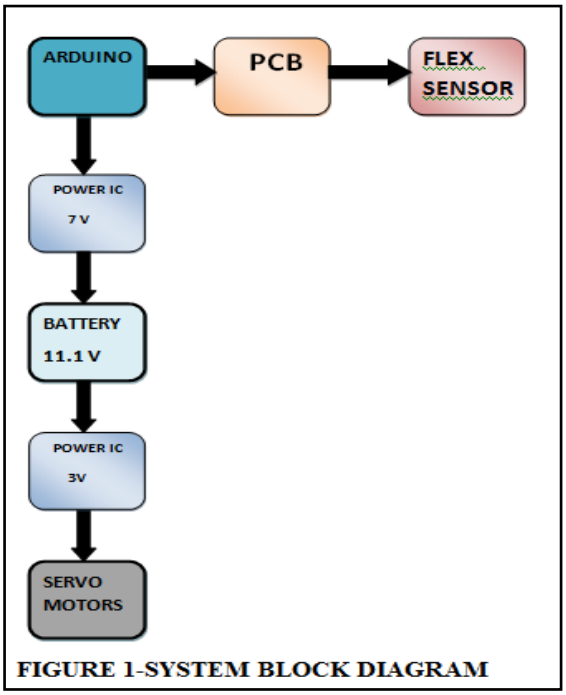


FIGURE 1-SYSTEM BLOCK DIAGRAM

This paper mainly analysis about different topologies and designs regarding the construction of this Arduino based animatronic hand. Although more complicated and precise(more expensive) versions of this concept have been developed, this is a fun project with many potential applications. Interactive robot control of this level, I think, has many uses in industrial manufacturing, medical research, and anything you want to be able to do with precision that is unsafe to touch.

The basic components of the hand and glove are the hand itself, the servos, the Arduino, the glove, and the flex sensors. The glove is mounted with flex sensors: variable resistors that change their value when bent. They're attached to one side of a voltage divider with resistors of a constant value on the other side. The Arduino reads the voltage change when the sensors are bent, and triggers the servos to move a proportional amount. The servo pull strings that act as tendons, allowing the fingers to move.

I. STATUS OF THE PROJECT

This project has been completed successfully, and my goal of integrating all of the underlying technologies has been met.

This animatronic hand is able to be controlled according to the controller's wish. It's capable of moving at the required degrees of freedom. It can also pick up things upto minimum desired weight . Now, we can use it as a shadow hand of ours which is of various use. With the help of sensors, this hand can now provide detailed telemetry, which can be exploited to generate innovative manipulation control systems or to provide detailed understanding of the external environment.

In the sections that follow, I will describe various outstanding areas that could stand to see some further refinement.

II. DESIGN

A. SYSTEM OVERVIEW

1. *WORKING PRINCIPAL OF ARDUINO HARDWARE-*

The Arduino microcontroller is an easy to use yet powerful single board computer that has gained considerable traction in the hobby and professional market. The Duemilanove board features an Atmel ATmega328 microcontroller operating at 5 V with 2 Kb of RAM, 32 Kb of flash memory for storing programs and 1 Kb of EEPROM for storing Parameters. The clock speed is 16 MHz, which translates to about executing about 300,000 lines of C source code per second. The board has 14 digital I/O pins and 6 analog input pins. There is a USB connector for talking to the host computer and a DC power jack for connecting an external 6-20 V power source, for example a 11.1 V battery, when running a program while not connected to the host computer. Headers are provided for interfacing to the I/O pins using 22 g solid wire or header connectors.

The Arduino programming language is a simplified version of C/C++. If you know C, programming the Arduino will be familiar. If you do not know C, no need to worry as only a few commands are needed to perform useful functions. An important feature of the Arduino is that we can create a control program on the host PC, download it to the Arduino and it will run automatically. Remove the USB cable connection to the PC, and the program will still run from the top each time you push the reset button. Remove the battery and put the Arduino board in a closet for six months. When you reconnect the battery, the last program you stored will run. This means that you connect the board to the host PC to develop and debug your program, but once that is done, you no longer need the PC to run the program.

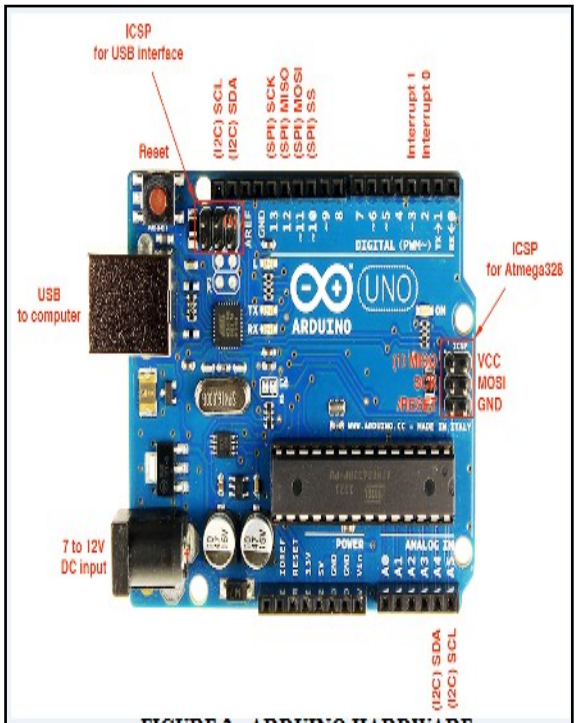


FIGURE 2- ARDUINO HARDWARE

The power of the Arduino is not its ability to crunch code, but rather its ability to interact with the outside world through its input-output (I/O) pins. The Arduino has 14 digital I/O pins labeled 0 to 13 that can be used to turn motors and lights on and off and read the state of switches. Each digital pin can sink or source about 40 mA of current. This is more than adequate for interfacing to most devices, but does mean that interface circuits are needed to control devices other than simple LED's. In other words, you cannot run a motor directly using the current available from an Arduino pin, but rather must have the pin drive an interface circuit that in turn drives the motor. A later section of this document shows how to interface to a small motor. To interact with the outside world, the program sets digital pins to a high or low value using C code instructions, which corresponds to +5 V or 0 V at the pin. The pin is connected to external interface electronics and then to the device being switched on and off. The sequence of events is shown in this figure.

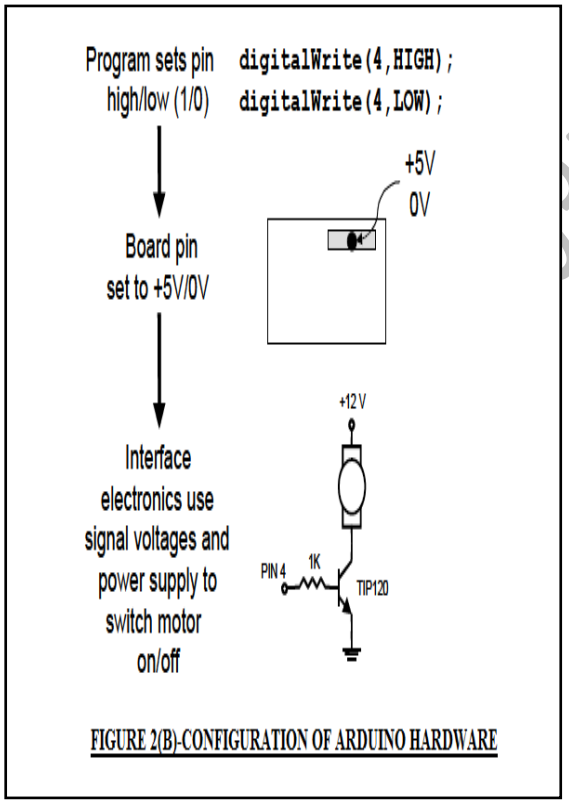


FIGURE 2(B)-CONFIGURATION OF ARDUINO HARDWARE

To determine the state of switches and other sensors, the Arduino is able to read the voltage value applied to its pins as a binary number. The interface circuitry translates the sensor signal into a 0 or +5 V signal applied to the digital I/O pin. Through a program command, the Ardiomp interrogates the state of the pin. If the pin is at 0 V, the program will read it as a 0 or LOW. If it is at +5 V, the program will read it as a 1 or HIGH. If more than +5 V is applied, you may blow out your board, so be careful. The sequence of events to read a pin is shown in this figure:

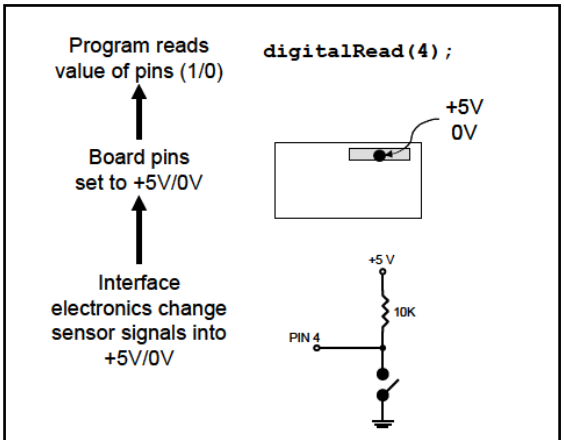


FIGURE 2(C): CONFIGURATION OF ARDUINO HARDWARE

Interacting with the world has two sides. First, the designer must create electronic interface circuits that allow motors and other devices to be controlled by a low (1-10 mA) current signal that switches between 0 and 5 V, and other circuits that convert sensor readings into a switched 0 or 5 V signal. Second, the designer must write a program using the set of Arduino commands that set and read the I/O pins.

When reading inputs, pins must have either 0 or 5V applied. If a pin is left open or "floating", it will read random voltages and cause erratic results. This is why switches always have a 10K pull up resistor connected when interfacing to an Arduino pin.

The reason to avoid using pins 0 and 1 is because those pins are used for the serial communications between the Arduino and the host computer. The Arduino also has six analog input pins for reading continuous voltages in the range of 0 to 5 V from sensors such as potentiometers.

2. ***WORKING PRINCIPAL OF SERVO MOTOR-***

Unlike dc motors, with servo motors you can position the motor shaft at a specific position (angle) using control signal. The motor shaft will hold at this position as long as the control signal not changed. This is very useful for controlling robot arms, unmanned airplanes control surface or any object that you want it to move at certain angle and stay at its new position. Servo motors may be classified according to size or torque that it can withstand into mini, standard and giant servos. Usually mini and standard size servo motors can be

powered by Arduino directly with no need to external power supply or driver.

The servo have 3 wires:

Black wire: GND (ground) !

RED wire:+5v !

Colored wire: control signal

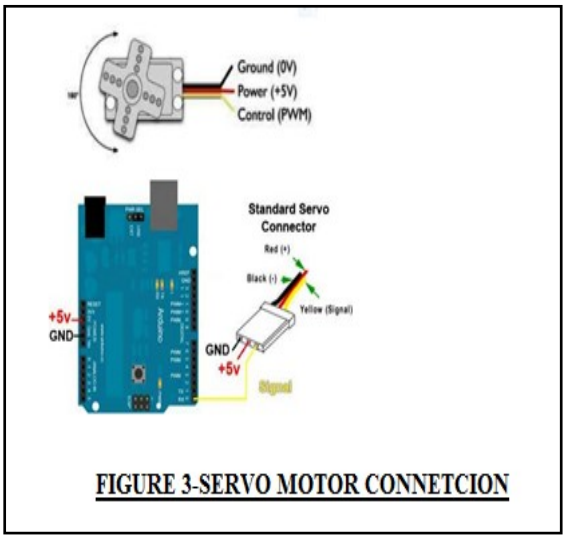


FIGURE 3-SERVO MOTOR CONNETCION

The third pin accept the control signal which is a pulse-width modulation (PWM) signal. It can be easily produced by all micro- controllers and Arduino board. This accepts the signal from your controller that tells it what angle to turn to. The control signal is fairly simple compared to that of a stepper motor. It is just a pulse of varying lengths. The length of the pulse corresponds to the angle the motor turns to.

The pulse width sent to servo ranges as follows:

Minimum: 1 millisecond ---> Corresponds to 0 rotation angle.

Maximum: 2 millisecond ---> Corresponds to 180 rotation angle.

Any length of pulse in between will rotate the servo shaft to its corresponding angle. For example : 1.5 ms pulse corresponds to rotation angle of 90 degree.

This is will explained in figure below:

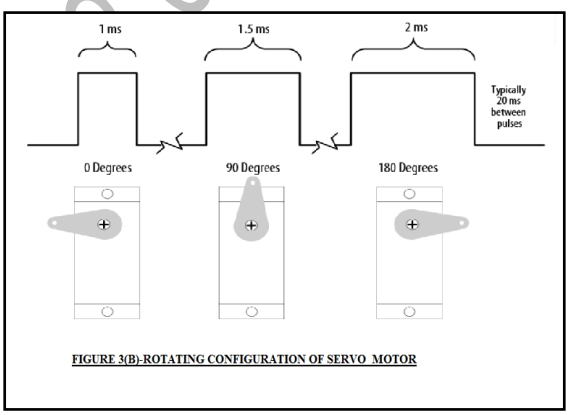


FIGURE 3(B)-ROTATING CONFIGURATION OF SERVO MOTOR

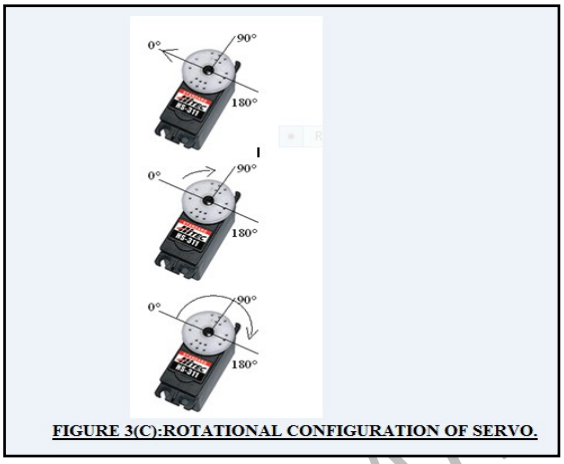


FIGURE 3(C):ROTATIONAL CONFIGURATION OF SERVO.

3.WORKING PRINCIPAL OF FLEX SENSOR-

Flex sensors are sensors that change in resistance depending how much the sensor is bend. Sensors convert the change in bend to electrical resistance - the more the sensor bend, the higher the resistance value. Using the Flex Sensor is very easy. There are couple of different manufacturers in the market.

Datasheet instructs you to use operational amplifier (opamps). That may be useful if you plan to use flex sensor as stand-alone device (without any microcontroller).Because We are using arduino, We skipped all OpAmps and made a very simple circuit with only one additional resistor.

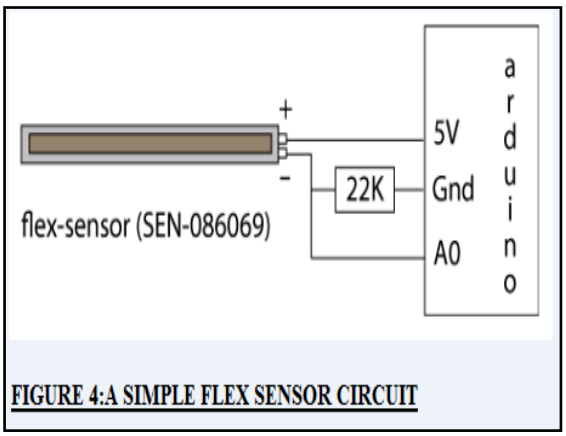


FIGURE 4:A SIMPLE FLEX SENSOR CIRCUIT

Varying the value of the resistor will results different readings. With 22k Ohm resistor I will get values between 300-700. This works fine for us. In our code we assumed that all values under 400 mean that the sensor is bend. All values above 600 mean that sensor is nor bend. Note that Flex sensor give reliable readings ONLY if you bend it on the specific direction (usually towards on the text side of the sensor).

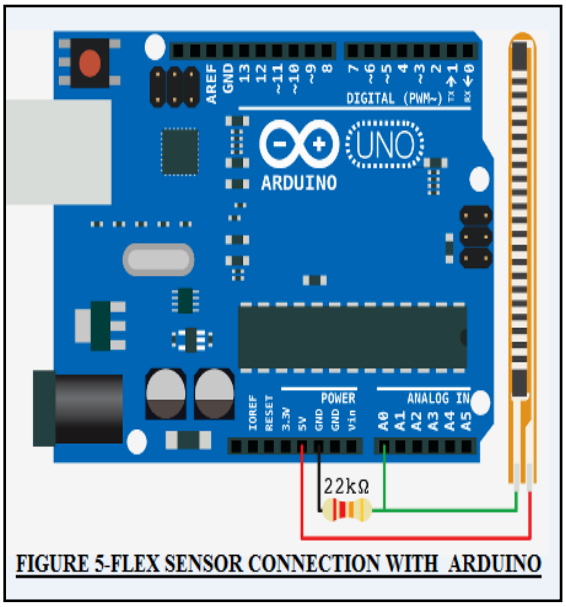


FIGURE 5-FLEX SENSOR CONNECTION WITH ARDUINO

4. *Motor Control Using Arduino-*

A. OPERATING ONE SERVO WITH ARDUINO:

Standard servo motor control using Arduino is extremely easy. This is because the Arduino software comes with a sample servo sketch and servo library that will get you up and running quickly:

1. Connect the black wire from the servo to the Gnd pin on the Arduino
2. Connect the red wire from the servo to the +5V pin on the Arduino
3. Connect the third wire (usually orange or yellow) from the servo to a digital pin on the Arduino.

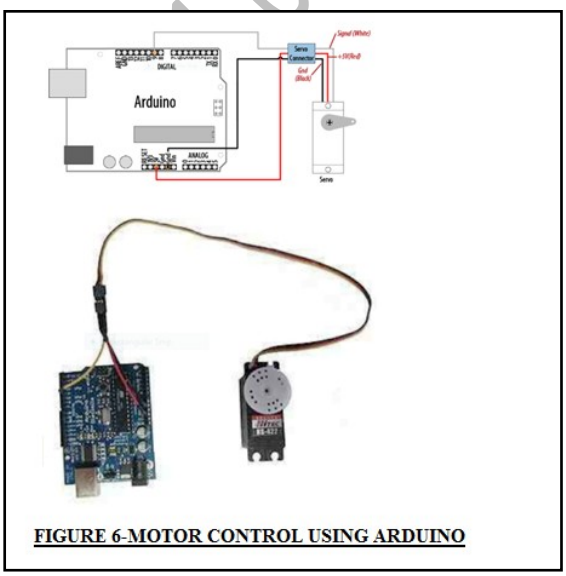


FIGURE 6-MOTOR CONTROL USING ARDUINO

 # Important Notes:

1- It is not a good idea to connect a motor of any kind directly to the Arduino because it usually requires more power than the board can provide.

2- In our example, the servo is being used to demonstrate code and is not encountering any resistance. Note that you should use a standard or small size. if you are uncertain, check the servo's no load current rating (it should usually be under 150mA).

3-You may need an external source of 5 or 6 volts when connecting multiple servos. Four AA cells work well if you want to use battery power. **Remember that you must connect the ground of the external power source to Arduino ground.**

B. OPERATING TWO SERVO WITH ARDUINO:

The Arduino can control two servos with the same ease as one. All it takes is creating a second instance (copy) of the Servo object, giving it a unique name. In this project we had to use six servos, five of which has to connect directly with arduino at the same process of connecting one servo with arduino. **When wiring the solderless breadboard, be especially careful not to mix positive and negative leads to the servo. Reversing the power will permanently damage it**.

In order for everything to function properly, the ground connections for the Arduino and the servo battery supply must be connected together.This is shown in both the schematic and pictorial circuit views. Make sure to also properly orient the connectors for the servos when you plug them into the board. Servo power leads are color-coded, but the colors aren't universal. Here two servos are being connected with the arduino in the figure given below:-

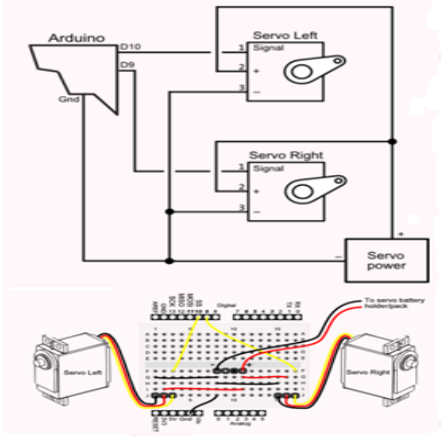


FIGURE 7-OPERATING TWO MOTORS USING ARDUINO.

C. SYSTEM ALGORITHM

1. ARDUINO PROGRAMMING CODING STYLE:-

The Arduino runs a simplified version of the C programming language, with some extensions for accessing the hardware. All Arduino instructions are one line. The board can hold a program hundreds of lines long and has space for about 1,000 two-byte variables. The Arduino executes programs at about 300,000 source code lines per sec. Programs are created in the Arduino development environment and then downloaded to the Arduino board. Code must be entered in the proper syntax which means using valid command names and a valid grammar for each code line. The compiler will catch and flag syntax errors.
before download. Sometimes the error message can be cryptic and you have to do a bit of hunting because the actual error occurred before what was flagged.
Style refers to our own particular style for creating code and includes layout, conventions for
using case, headers, and use of comments. All code must follow correct syntax, but there are many different styles we can use. Here are some suggestions:

- Starting every program with a comment header that has the program name and perhaps a brief description of what the program does.

- Using indentation to line things up. Function name and braces are in column one. Mark major sections or functions with a comment header line or two.

- Having just the right number of comments, not too few and not too many. Assume the reader knows the programming language so have the comment be instructive. Here is an example of an instructive comment

```
digitalWrite(4,HIGH) // turn on
motor
```

and here is a useless comment

```
digitalWrite(4,HIGH) // set pin 4
HIGH
```

we need not comment every line. In fact, commenting every line is generally bad practice.

- Adding the comments when you create the code. If you tell yourself, "Oh, I'll add the comments when the code is finished", you will never do it.

2.THE PROCESSING CODE:-

The processing code of our project is given below:-

```
#include <Servo.h>

int flexSensorPin1 = A0;
int flexSensorPin2 = A0;
int flexSensorPin3 = A0;
int flexSensorPin4 = A0;
int flexSensorPin5 = A0;
int flexSensorPin6 = A0;

Servo
servo1,servo2,servo3,servo4,ser
vo5,servo6;

void setup(){
Serial.begin(9600);
servo1.attach(9);
servo2.attach(8);
servo3.attach(7);
servo4.attach(6);
servo5.attach(5);
servo6.attach(4);
pinMode(flexSensorPin1,INPUT);
pinMode(flexSensorPin2,INPUT);
pinMode(flexSensorPin3,INPUT);
pinMode(flexSensorPin4,INPUT);
pinMode(flexSensorPin5,INPUT);
pinMode(flexSensorPin6,INPUT);

}
int pos=0;
void loop(){
int
fsr1=analogRead(flexSensorPin1)
;
//myServo.write(x);
//Serial.println(fsr);
int s1=map(fsr1, 605, 696, 180,
0);
Serial.println(s1);
if(s1>0 ||s1<180)
servo1.write((s1));
delay(100);
int
fsr2=analogRead(flexSensorPin2)
;
//myServo.write(x);
//Serial.println(fsr);
int s2=map(fsr2, 605, 696, 180,
0);
Serial.println(s2);
if(s2>0 ||s2<180)
servo2.write((s2));
delay(100);
int
fsr3=analogRead(flexSensorPin3)
;
//myServo.write(x);
```

```
//Serial.println(fsr);
int s3=map(fsr3, 605, 696, 180,
0);
Serial.println(s3);
if(s3>0 ||s3<180)
servo3.write((s3));
delay(100);
int
fsr4=analogRead(flexSensorPin4)
;
//myServo.write(x);
//Serial.println(fsr);
int s4=map(fsr4, 605, 696, 180,
0);
Serial.println(s4);
if(s4>0 ||s4<180)
servo4.write((s4));
delay(100);
int
fsr5=analogRead(flexSensorPin5)
;
//myServo.write(x);
//Serial.println(fsr);
int s5=map(fsr5, 605, 696, 180,
0);
Serial.println(s5);
if(s5>0 ||s5<180)
servo5.write((s5));
delay(100);
int
fsr6=analogRead(flexSensorPin6)
;
//myServo.write(x);
//Serial.println(fsr);
int s6=map(fsr6, 605, 696, 180,
0);
Serial.println(s6);
if(s6>0 ||s6<180)
servo6.write((s6));
delay(100);
```

3.THE CODING STRUCTURE ANALYSIS:

In the program above, the very first thing that we did in the setup function is to begin serial communications, at 9600 bits of data per second, between our Arduino and our computer with the line:

```
Serial.begin(9600);
```

Next, initialize digital pin 8, the pin that will read the output from your button, as an input:

```
pinMode(A8,INPUT);
```

Now let's move into the main loop of your code.

A. FUNCTIONS:

1.ARDUINO PROGRAM FUNCTION

All Arduino programs have two functions, setup() and loop().Other functions must be created outside the brackets of those two functions. The instructions you place in the startup() function are executed once when the program begins and are used to initialize. Use it to set directions of pins or to initialize variables. The instructions placed in loop are executed repeatedly and form the main tasks of the program. Therefore every program has this structure:

```
void setup()
{
// commands to initialize go here
}
void loop()
{
// commands to run your machine go
here
}
```

The absolute, bare-minimum, do-nothing program that we can compile and run is

```
void setup() {} void loop() {}
```

The program performs no function, but is useful for clearing out any old program. Note that the compiler does not care about line returns, which is why this program works if typed all on one line.

2.Void() : The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

3.SERVO CONTROL PROGRAMMING:

There are two main portions of code that you'll see explained and in detail below:

-Initialization
-Servo Control

The first part of the code shows you how to initialize the servo and all our variables, this is important if you want to use more than 1 servo you need to declare that. We'll be using the Arduino Servo library for all out control to make things as easy as possible.

Initialization

----------« Begin Code »----------

```
#include <Servo.h>

Servo myservo;  // create servo
object to control a servo
                // a maximum of
eight servo objects can be created

void setup()
{
  myservo.attach(0);  // attaches
the servo on pin 0 to the servo
object
```

```
  delay(5000);
}
..
...
```

----------« End Code »----------

The last portion of the code is where we actually tell the servo where to move using the write function of the Arduino's Servo library.

Main Loop

----------« Begin Code »----------

```
...
..
void loop()
{
 //Move To 0 Degrees
myservo.write(10);
 delay(5000);

 //Move To 45 Degrees
 myservo.write(45);
 delay(5000);

 //Move To 90 Degrees
 myservo.write(90);
 delay(5000);

//Move To 135 Degrees
 myservo.write(135);
 delay(5000);
 //Move To 180 Degrees
 myservo.write(170);
 delay(5000);

 //Move To 135 Degrees
 myservo.write(135);
 delay(5000);

 //Move To 90 Degrees
  myservo.write(90);
  delay(5000);
  //Move To 45 Degrees
  myservo.write(45);
  delay(5000);

}
```

----------« End Code »----------

B. The Simple Commands:-

This section covers the small set of commands we needed to make the Arduino being operated properly.

1. Serial.begin()

Description: Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates**: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200**. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. An optional second argument configures the data, parity, and stop bits. The default is 8 data ***bits, no parity, one stop bit.***

Syntax:

```
Serial.begin(speed)
Serial.begin(speed, config)
```

Arduino Mega only:

```
Serial1.begin(speed)
Serial1.begin(speed, config)
```

Parameters:

speed: in bits per second (baud) – *long.*

2. servo.attach()

Description:-Attach the Servo variable to a pin. Note that in Arduino 0016 and earlier, the Servo library supports only servos on only two pins: 9 and 10.

Syntax:`servo.attach(pin)`
`servo.attach(pin, min, max)`

Parameters:***servo***: a variable of type Servo pin: the number of the pin that the servo is attached to

min (optional): the pulse width, in microseconds, corresponding to the minimum (0-degree) angle on the servo (defaults to 544)

max (optional): the pulse width, in microseconds, corresponding to the maximum (180-degree) angle on the servo (defaults to 2400).

3. pinMode()

Description: Configures the specified pin to behave either as an input or an output.This command, which goes in the setup() function, is used to set the direction of a digital I/O pin. Set the pin to OUTPUT if the pin is driving and LED, motor or other device. Set the pin to INPUT if the pin is reading a switch or other sensor. On power up or reset, all pins default to inputs.

Syntax: pinMode(pin, mode)

Parameters:

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT,

Returns : None

4. serial.print()

Description: The Serial.print command lets you see what's going on inside the Arduino from your computer.
For example, we can see the result of a math operation to determine if you are getting the right
number. Or, we can see the state of a digital input pin to see if the Arduino is a sensor or switch
properly. When our interface circuits or program does not seem to be working, use the Serial.print command to shed a little light on the situation. For this command to show anything,
you need to have the Arduino connected to the host computer with the USB cable.
For the command to work, the command Serial.begin(9600) must be placed in the setup()
function. After the program is uploaded, we must open the Serial Monitor window to see the
response.
There are two forms of the print command. **Serial.print() prints** on the same line while **Serial.println()** starts the print on a new line.
Here is a brief program to check if your board is alive and connected to the PC

```
void setup()
{
Serial.begin(9600);
Serial.println("Hello World");
}
void loop() {}
```

Here is a program that loops in place, displaying the value of an I/O pin. This is useful for
checking the state of sensors or switches and to see if the Arduino is reading the sensor properly.
Try it out on your Arduino. After uploading the program, use a jumper wire to alternately connect pin 2 to +5V and to Gnd.

```
void setup()
{
Serial.begin(9600);
}
void loop()
{
Serial.println(digitalRead(2));
delay(100);

}
```

If we wanted to see the states of pins 2 and 3 at the same time, you can chain a few print commands, noting that the last command is a println to start a new line.

```
void setup()
{
Serial.begin(9600);
}
void loop()
{
Serial.print("pin 2 = ");
Serial.print(digitalRead(2));
Serial.print(" pin 3 = ");
Serial.println(digitalRead(3));
21
delay(100);

}
```

5. **analogRead()**

   Description: Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax : analogRead(pin)

Parameters : pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns:- int (0 to 1023)

6. **map(value, fromLow, fromHigh, toLow, toHigh)**

Description: Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc. Does not constrain

values to within the range, because out-of-range values are sometimes intended and useful. The constrain() function may be used either before or after this function, if limits to the ranges are desired. Note that the "lower bounds" of either range may be larger or smaller than the "upper bounds" so the map() function may be used to reverse a range of numbers, for example

```
y = map(x, 1, 50, 50, 1);
```

The function also handles negative numbers well, so that this example

```
y = map(x, 1, 50, 50, -100);
```
is also valid and works well.

The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged.

Parameters :

value: the number to map

fromLow: the lower bound of the value's current range

fromHigh: the upper bound of the value's current range

toLow: the lower bound of the value's target range

toHigh: the upper bound of the value's target range

Returns: The mapped value.

7. If

Description: This is the basic conditional branch instruction that allows your program to do two different things depending on whether a specified condition is true or false. If the condition is true, the program will execute the commands between the braces. If the condition is not true, the program will skip to the statement following the braces.The condition compares one thing to another.

delay()

Description: Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.) Since most interactions with the world involve timing, this is an essential instruction. The delay can be for 0 to 4,294,967,295 msec. In the project we used

delay(100); // waits 50 ms

Syntax:delay(ms)
Parameters:

ms: the number of milliseconds to pause (*unsigned long*)

Returns: nothing.

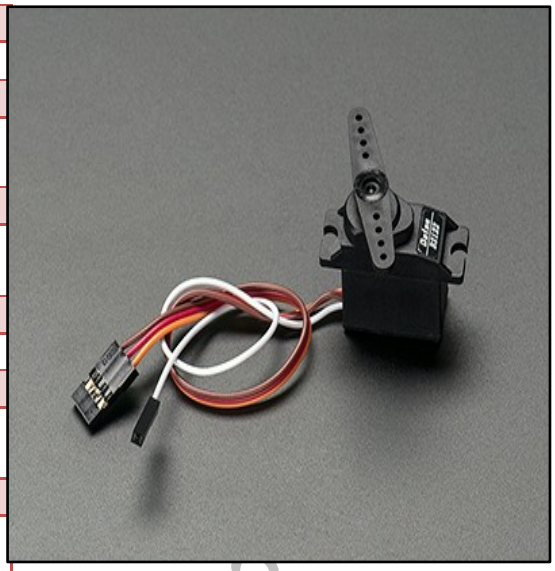
III. *SPECIFICATION*

A. HARDWARE DETAILS

PARTS LIST:

COMPONENTS	SPECIFICATION	QUANTITY (Unit)
1. ARDUINO	MEGA 2560	1
2. SERVO MOTOR	SG90 (2.5K)	3
3. SERVO MOTOR	MG996R(10K)	2
4. SERVO MOTOR	SMS4315 (14K TORQUE)	1
5. BREAD BOARD	SOLDER LESS	1
6. JUMPER WIRE	MALE, FEMALE , MALE-FEMALE	3
7. FLEX SENSOR	4.5”	5
8. BATTERY	11.1V	1
9. BATTERY CHARGER	N/A	1
10. CHARGER CONNECTING CABLE	N/A	1

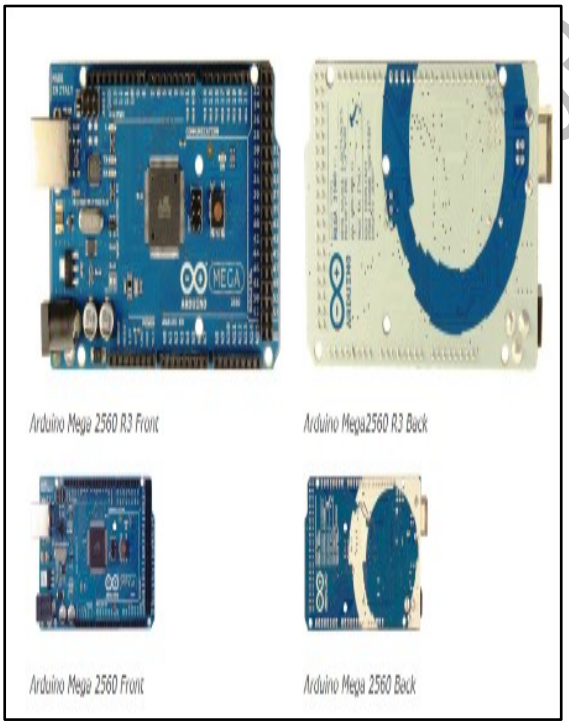
11. POWER IC	N/A	2
12. PCB	N/A	2
13. RESISTORS	22K	5
14. POT	10K , 100K	2
15. STRING	N/A	5
16. NEEDLE & THREAD	N/A	1
17. SUPER GLUE	N/A	1
18. A POWER DRILL	N/A	1
19. SPRING	N/A	10
20. RIGHT HAND GLOVE	N/A	1
21. WOOD	N/A	2
22. PLASTIC HUIS PIPES	N/A	4

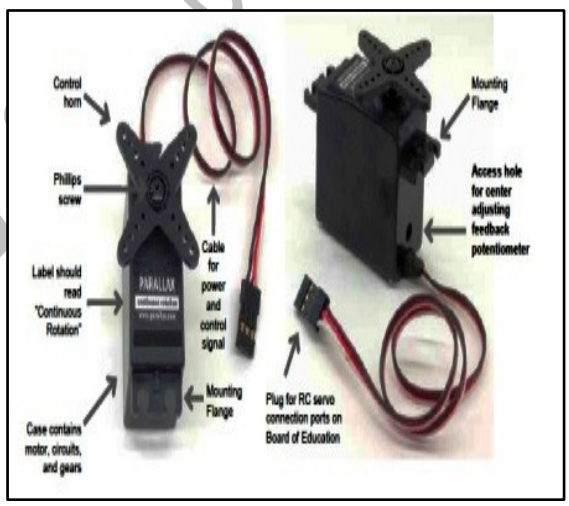


3. **SERVO MOTOR 14K TORQUE**

The associated pictures related to the hardware components are given below:-

1. **ARDUINO MEGA 2560**

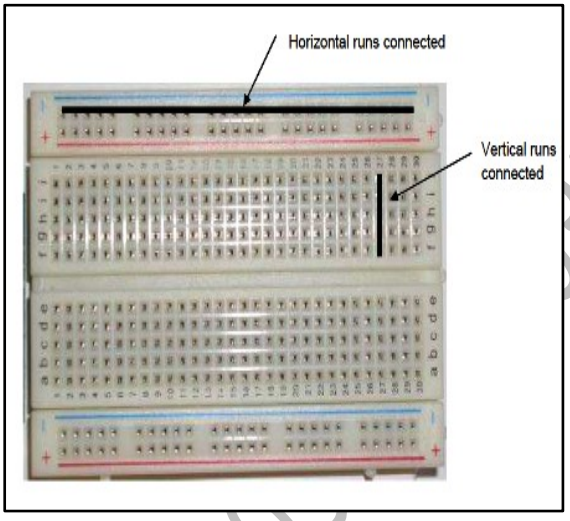




2. **SERVO MOTOR SG90**

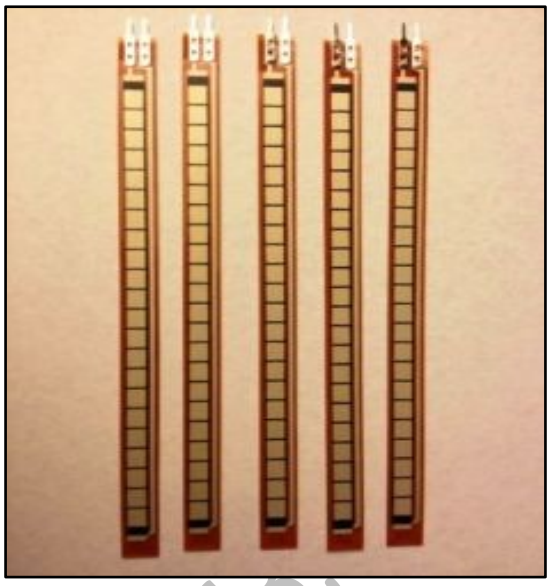
4. **SERVO MOTOR 10K TORQUE**

5. **BREAD BOARD**



6. **JUMPER WIRE**

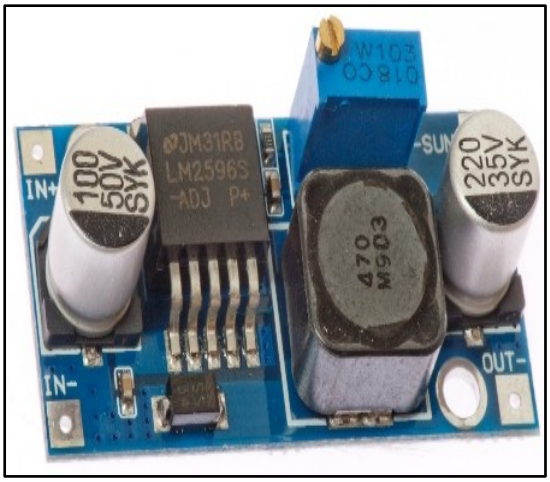
7. **FLEX SENSOR**



8. **11.1V BATTERY**

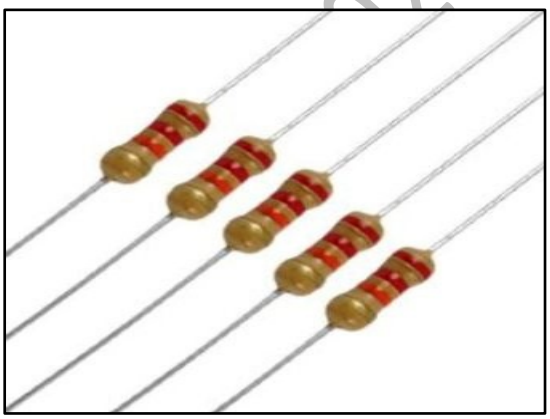
9. **BATTERY CONNECTOR**

10. **POWER IC**



9.POTENTIOMETER (10K,100K)

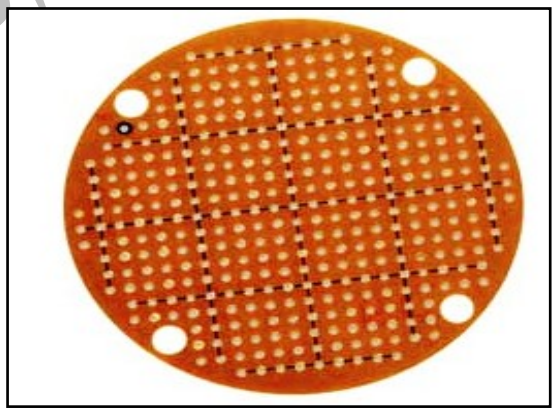
9. **22K RESISTORS**



10. **SPRING**

11. **STRING**

12. **PCB**



IV. IMPLEMENTATION AND CONSTRUCTION

A. HARDWARE PROCEDURE

1.Set up the Sensor Circuit

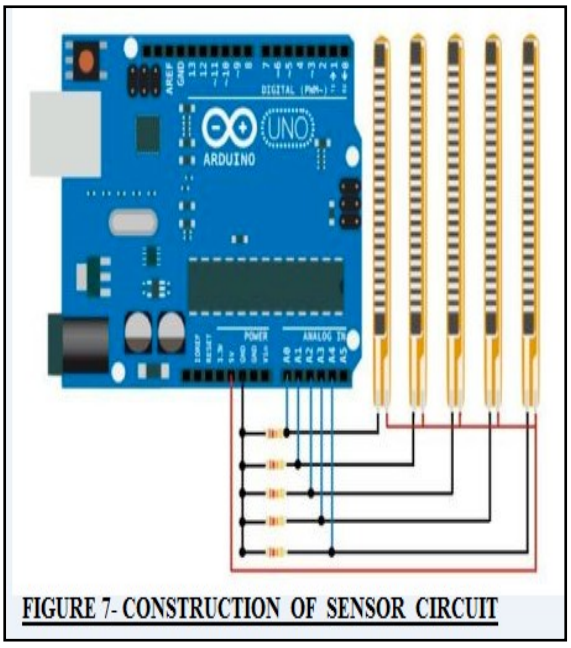


FIGURE 7- CONSTRUCTION OF SENSOR CIRCUIT

The flex sensors require a circuit in order for them to be compatible with Arduino. It's a voltage divider: the flex sensors are variable resistors, and when paired with resistors of a static value, change in resistance (in this case bending the sensor) can be sensed through the change in voltage between the resistors. This can be measured by the Arduino through its analog inputs. The schematic is attached (red is positive voltage, black is negative, and blue goes to the Arduino). The resistors in the photo are 22K. I color-coded the wires we used in the same way as the schematic, so we can see more easily.
The main GND wire, which is connected to all the individual GND wires from the sensors, gets plugged into the Arduino's GND. The +5V from the Arduino goes to the main positive voltage wire, and each blue wire gets plugged into a separate analog input pin.

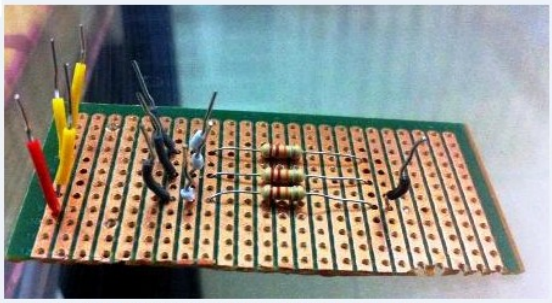


FIGURE 7(B)- SOLDIERED THE SENSOR CIRCUIT ON A PCB.

Then we soldered the circuit onto a small PCB.One that could be easily mounted onto the glove.

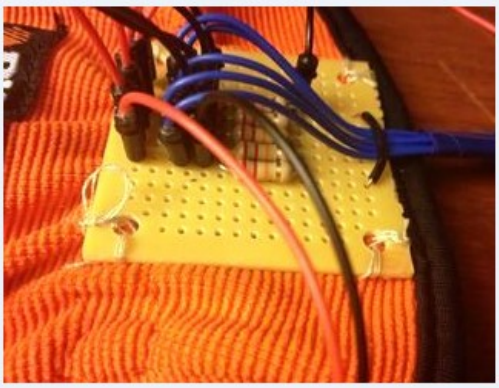


FIGURE 9-PCB MOUNTED ON THE GLOVES

We were able to solder the wires to the sensors relatively easily also, and used heat shrink to make sure there were no shorts.

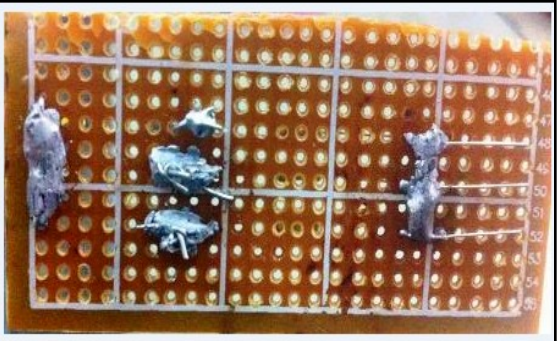


FIGURE 7(C)-BACK OF THE PCB(SHORT PREVENTED).

We then wrapped the area where the wires are connected to the sensors with electrical tape to stabilize the sensors.

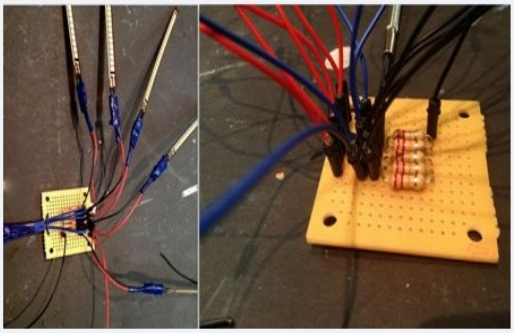
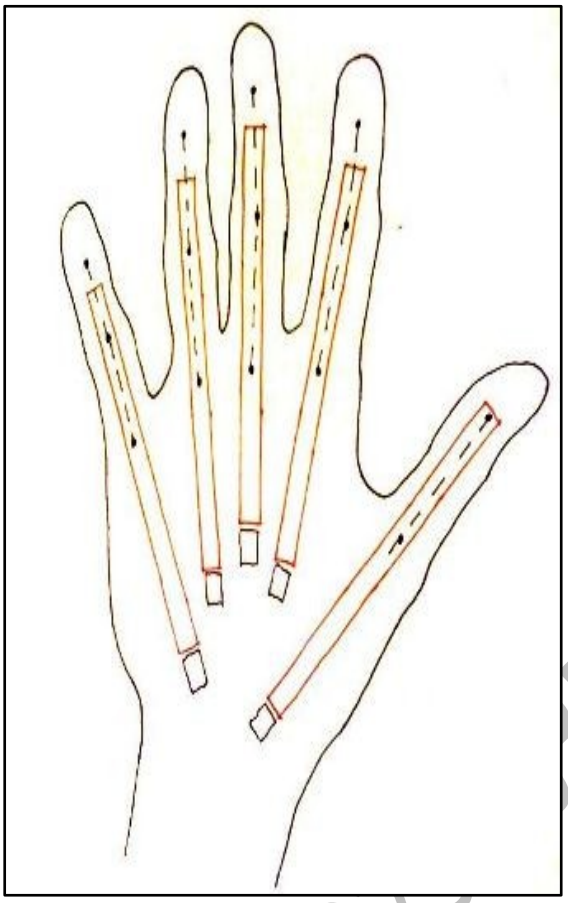


FIGURE 9 : CONSTRUCTION OF SENSOR CIRCUIT

Near the bottom, where the leads are attached, the sensors are a bit weaker and the tape ensures that they won't bend too far and won't get damaged.

3. FLEX SENSOR MOUNTING



Now it's time to mount the sensors and their circuit onto the glove itself. First, We drilled a tiny hole in the plastic of the sensors (at the top, once the resistive material has ended). Be sure not to hit the resistive material! Then, put on the glove and pull it tightly to your hand. On each finger, with a pencil or pen, make small lines over the tops of each joint/knuckle. This will tell you where to sew the sensors. Sew each sensor tip to the area of each finger just above where each of your fingernails would be (use the hole you just drilled). Then, for each sensor, make loose loops around them with thread at both joints in each finger. Once each sensor is in place and slides under the loops of thread nicely.

FIGURE 8- FLEX SENSOR MOUNTING ON THE GLOVE

Then We sewed the PCB onto the wrist part of the glove tightly.

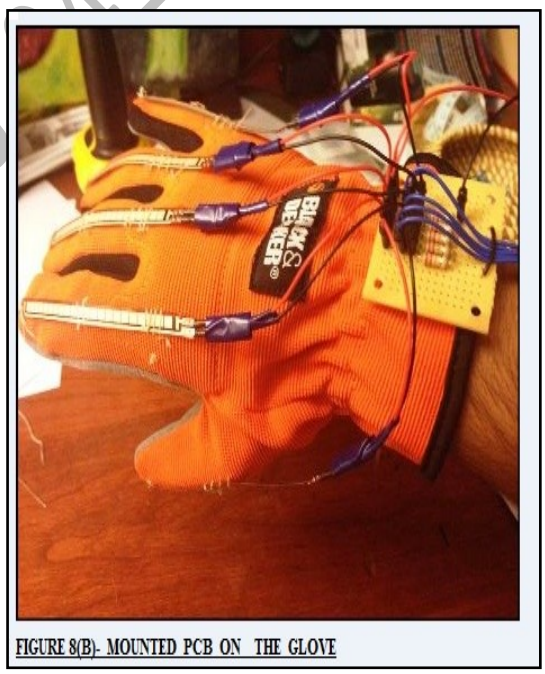


FIGURE 8(B)- MOUNTED PCB ON THE GLOVE

REMEMBER: for each step in this process, be sure We're not sewing the glove itself closed. That's quite a hassle.

4. HAND AND SERVO BED CONSTRUCTION

1. FINGERS ASSEMBLY

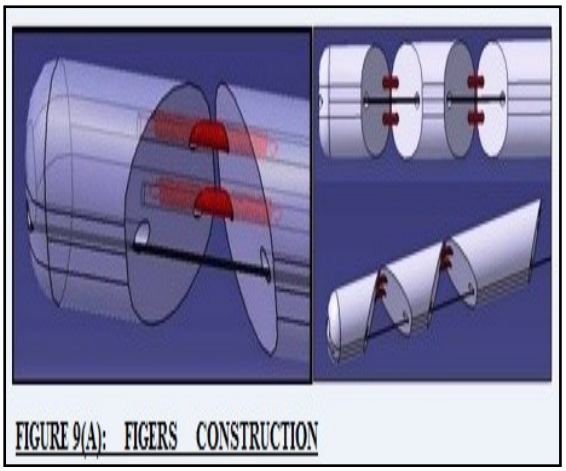
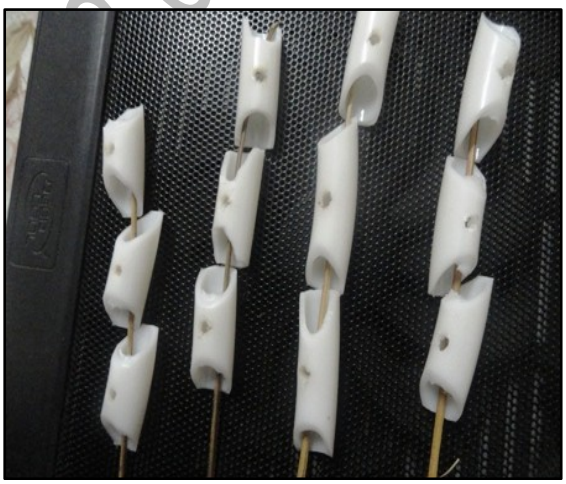
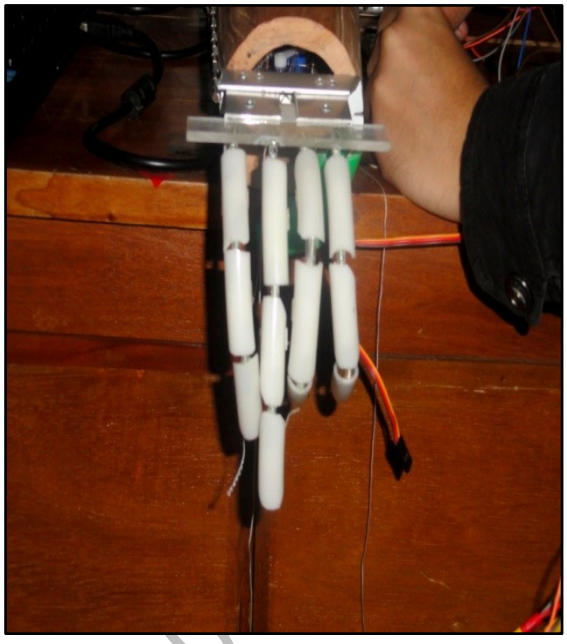


FIGURE 9(A): FIGERS CONSTRUCTION

We used plastic hollow pipes to construst the finger parts.When assembling the fingers, we made sure that the parts were oriented correctly before gluing. Also, we made make sure to re-drill the holes on the finger parts so the 3mm screws will act as hinge pins without causing friction.Then we connected the part using a string and screws.

FIGURE 9(B) :FINGERS ASSEMBLY.





Then we kept the screws in with a dab of hot glue on the outside of the fingers.

2. SERVO BED CONSTRUCTION AND SERVO MOUNTING:-

We have used a wooden piece to make the servo bed. Here in the bed we made 5 servo motors to be held perfectly within the bed. It's just kind of a bed for the servos. Later we have to put the strings to connect the servos with each other. Here is the construction figure shown below:-

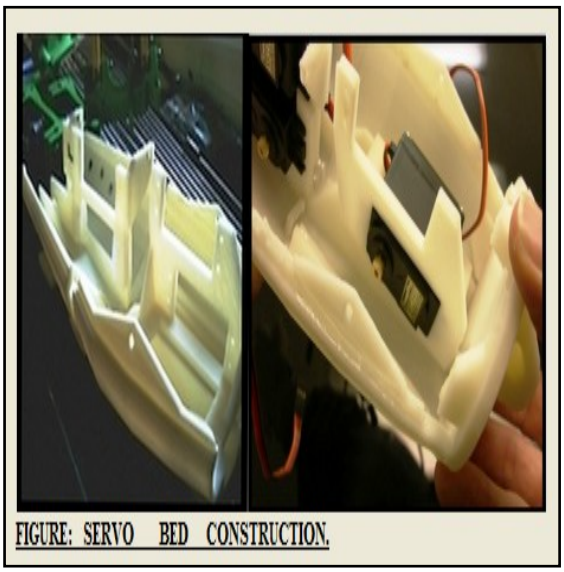


FIGURE: SERVO BED CONSTRUCTION.

Here is the picture of our wooden aervo bed shown below:-

FIGURE 10- WOODEN CONSTRUCTED SERVO BED.

We waited before installing the strings into the hand as we wanted to make sure the servos are working first.

3. ADDING THE STRINGS:

Adding the strings is by far the hardest and most tedious part of this project. It's simple in concept, but difficult to actually execute. Threading the fingers takes patience: remember that. The one difference between my installation of the strings and I used hot glue. To me, hot glue is more adjustable when calibrating each finger because it can be easily melted and re-hardened. We connected the servo motor in a way we could move our fingers with the exact comfort and flexilbity. For that we calibrated the servo motors to connect the strings with the exact process

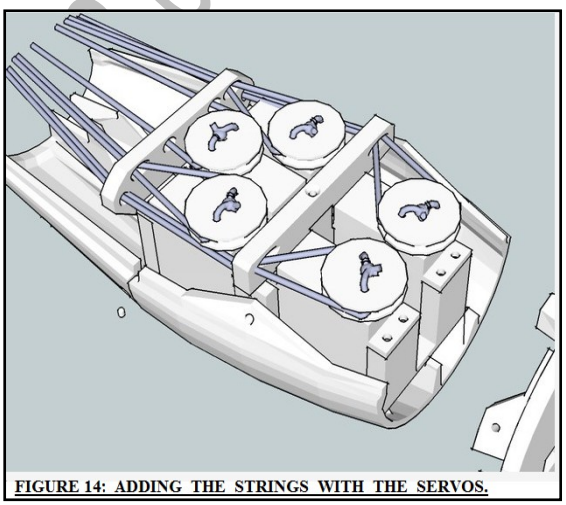


FIGURE 14: ADDING THE STRINGS WITH THE SERVOS.

To calibrate each servo ring so it flexes and relaxes its finger when we wanted it to based on the input, first we plugged in our Arduino and servo battery and run the program. Then we put the glove on and flex the finger that corresponds to the servo we're working on. We adjusted the servo ring so one hole is closest possible to the fingers and pulled the "relax" string of that finger as tightly as we can without bending the finger. We put it through the closest hole of the ring and glue it in place. Then, straightened my finger and pulled and secured the other string into the other hole. Then we repeated this process with each finger. It's important to make each string taut.

4. SETTING UP THE SERVO MOTOR CIRCUIT:

1. SERVO MOTOR WIRING WITH THE ARDUINO:

We connected the 5 servo motors (which had been mounted on the servo-bed) and the another servo (mounted on the hinge portion) with the ARDUINO PWM pins. The PWM input wire (orange) for each servo would be attached to the Arduino pins as follows:

- Servo One - PWM -> Digital 9,
- Servo Two - PWM -> Digital 8,
- Servo Three - PWM -> Digital 7,
- Servo Four - PWM -> Digital 6,
- Servo Five - PWM -> Digital 5,
- Servo Six - PWM -> Digital 4,

Following is the diagram showing the wiring of 6 servo motors with the Digital (PWM) pins of ARDUINO:-

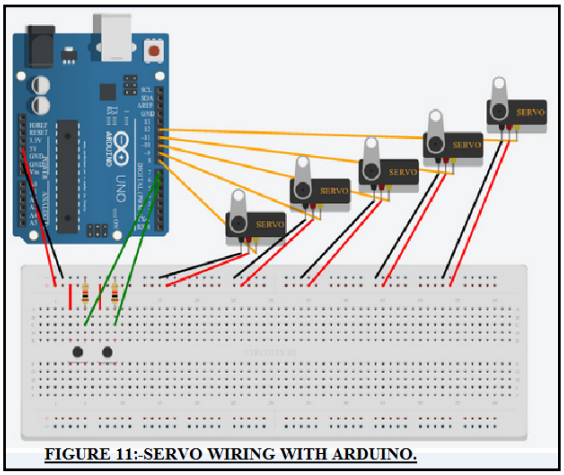


FIGURE 11:-SERVO WIRING WITH ARDUINO.

1. WIRING OF THE SIXTH (WRIST) SERVO MOTOR:

The servo motor which we have mounted on the wrist for it's movement .Then it's control(orange) wire was first connected with the PWM pin 4.

Servo Six - PWM -> Digital 4

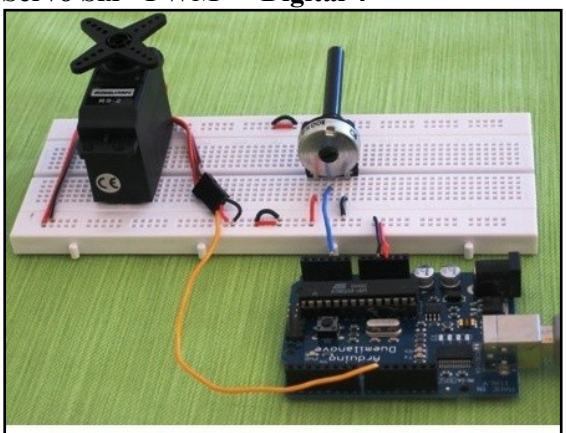


FIGURE 11(C)-WIRING OF THE SIXTH SERVO WITH ARDUINO

After that we connected its red and black (power) wire of the servo with a small solder-less bread board. We also connected a Potentiometer in series with the power wires. We connected the positive and the ground wire of the POT in series with the power wires of the servo and connected the variable resistive wire of the POT with an analog pin of the ARDUINO.

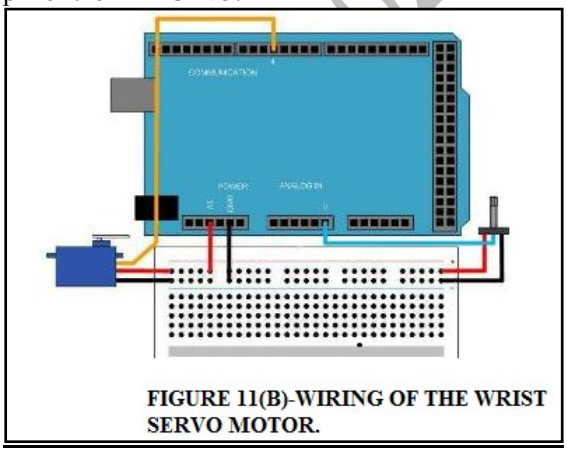


FIGURE 11(B)-WIRING OF THE WRIST SERVO MOTOR.

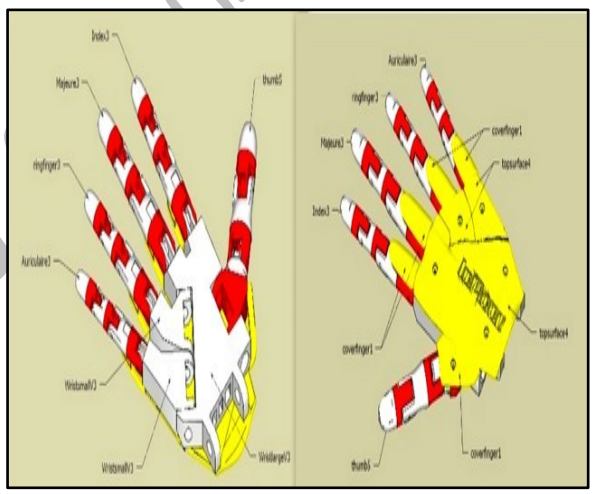
5. THUMP CONSTRUCTION:

We used aluminium plate and a piece of plastic glass for the thump construction. The larger portion of t he wrist(Top surface) has been constructed with aluminium plate and has been threaded using threading machine. The rest of the wrist has been constructed using a plastic glass plate with which we assembled the 5 fingers using super glue and screws keeping the comfortable position of the fingers in mind.

5. JOINING THE FINGERS WITH THE THUMP:

We assembled the 5 fingers with the plastic glass made portion of the wrist using super glue and screws.

According to the following pictures we tried to join all the fingers keeping comfort area and the top and bottom view of the fingers in mind.



5.BREAD BOARD DETAIL:

We used a bread board here to construct a particular circuit for the operation of 16 k torque servo motor for the control over the total hinge of the hand.

V. FUCTIONAL TESTING

A. SOFTWARE IMPLEMENTATION

1. SOFTWARE DOWNLOAD:

Following the instructions on the Getting Started section of the Arduino web site, http://arduino.cc/en/Guide/HomePage. we at first downloaded the latest

version of Arduino, **arduino-1.0.5-windows.**

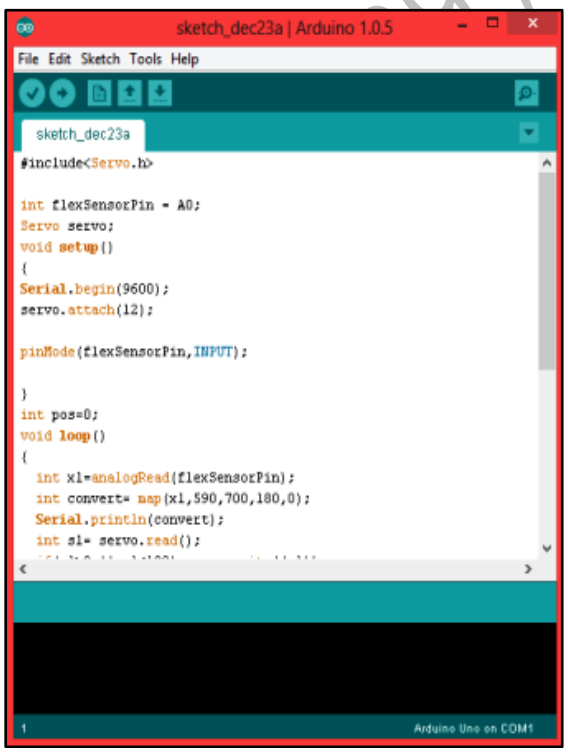
2. SOFTWARE INSTALLATION:

Going all the way through the steps to where we saw the pin 13 LED blinking. This is the indication that we had all software and drivers successfully installed and could start exploring with our own programs.

3. SOFTWARE EXECUTION:

1.Connected our Arduino to the computer with the USB cable. We did not need the battery for that time.
The green PWR LED will light. If there was already a program burned into the Arduino, it will
run.

2.Started the Arduino development environment. In Arduino-speak, programs are called "sketches",
but here we will just call them programs.
Our window would look something like this:-

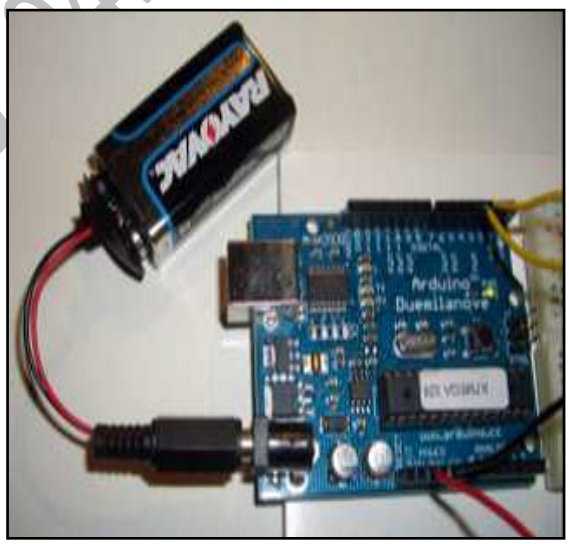


3.Then we clicked the Upload button or Ctrl-U to compile the program and load on the Arduino board.

B. POWER ISSUE:

1. CONNETCING A BATTERY:

1.For stand-alone operation, the board is powered by a battery rather than through the USB connection to the computer. While the external power can be anywhere in the range of 6 to 24 V
(We used here a 11.1V Lithium ion battery), a standard 9 V battery is convenient.
2.We then stablished Vin and Gnd connections on the board, it is better to solder the battery snap leads to a DC power plug and connect to the power jack on the board.



N.B Watch the polarity as you connect your battery to the snap as reverse orientation could blow out your board.
3. Then we disconnected our Arduino from the computer. Then we connected a 11.1 V battery to the Arduino power jack using the battery snap adapter confirming that the blinking program runs. This showed that we could power the Arduino from a battery and that the program we download ed runs without needing a connection to the host PC.

2. SETTING UP THE VOLTAGE OF POWER IC:

Battery power input is 11.1V. This input is supplied to 2 power ICs. Through power ICs we have controlled the input voltage to get the desired output of 7V & 3V. As the threshold voltage of arduino is 7V and the threshold voltage of servo is 3V.One power IC is set to get output of 7V. The output terminal of that power IC is connected to Vin and Gnd pin of arduino. As the threshold voltage of servo motor is 3V.Here the power adjustment of the power IC is done by keeping the multimeter positive and negative terminals into the input side of the power IC, IN+ & IN- and we varied the voltage with the help of power IC voltage-variable nob. Another power IC is set to get output of 3V. The output terminal of that power IC is connected to PCB and 6 terminals are made short with that so that 6 motors can get power supply.

Here the voltage adjustment diagram of power IC is given below.

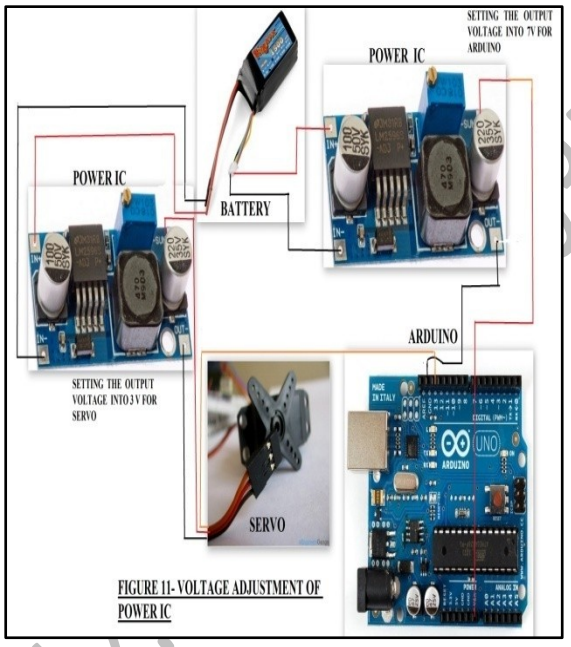


FIGURE 11- VOLTAGE ADJUSTMENT OF POWER IC

4. CALIBRATING THE FLEX SENSORS:

The sensor is simply a variable resistor that changes its resistance in response to how much it is bent. The way that the device works is based on its construction. The sensor has a backplane of a thin plastic polymer that can easily bend.

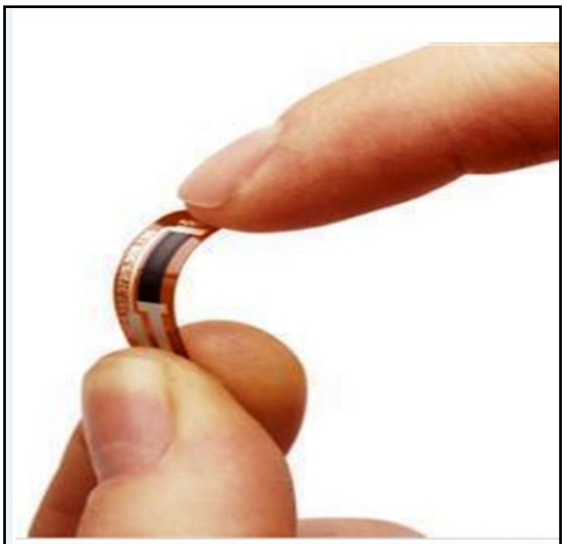
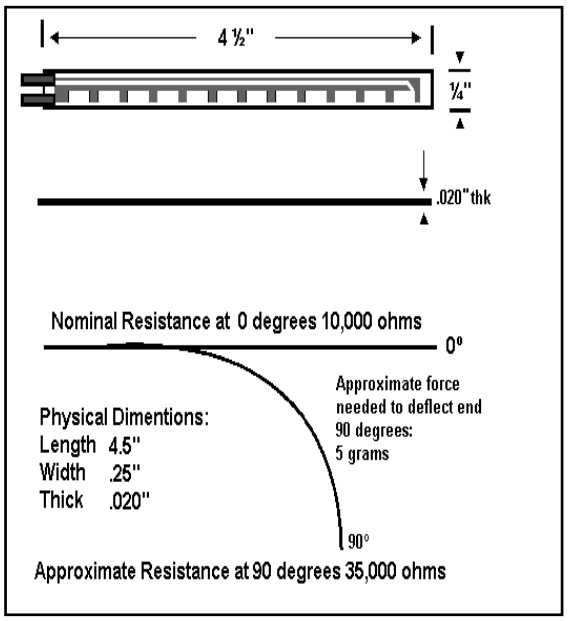


FIGURE 12- CALIBRATING THE FELX SENSORS.

The bend sensor consists of a coated substrate, such as plastic, that changes in electrical conductivity as it is bent. This provides non-mechanical reliability in electronic sensing and actuator technology.On top of this polymer is a layer of carbon that forms a resistor (which connects between the two metal terminals on the left side).On top of the carbon are equally sized and spaced squares of metal which are glued in place at their centers to the backplane. When the device is bent the spaces between the metal plates grows larger and thus the amount of carbon resistor shorted out by each metal square is reduced. When that happens the resistance of the sensor goes up. The response of the sensor is very linear over its flexible range. The sensor can only reliably indicate bending in one direction (into the page as it is pictured here).



For calibrating the sensors, we used the sensor with ARDUINO MEGA we fed the +5volt D.C as we had a lot of problems getting reliable readings from the sensor at first.. power supply voltage from ARDUINO into one side of the sensor and then measured the voltage on the other side of the sensor using the yellow "analog in" jack on ARDUINO. Since the device is simply a variable resistor the voltage drop across the resistor changes with the amount of bend the sensor undergoes.To calibrate the sensor we simply created "our own sensor" profile under ARDUINO programming software and used the on screen volt meter to create a two point calibration at zero and at 50% (90 degrees) of bend.

5. TESTING THE SERVOS:

At this point, the servos should already be mounted into the forearm. To connect them to the power supply and Arduino, We used a small solder-less breadboard. Connected each positive wire of the servo (usually red) to one of the rails on the breadboard, and the negative side of them (usually black or brown) to another rail. I should be remembered to connect the negative rail on the breadboard to the Arduino's other GND: all the GNDs in a circuit need to be connected for it to work. The +VCC can be different, but the GNDs need to be the same.

FIGURE 13- TESTING THE SERVOS.

Then we uploaded the program to the Arduino (the file is attached) and made sure of all the connections to the glove and servos are correct. Put on the glove, and turned on the Arduino. The servos should rotate based on how much our fingers are bent. If this is the case, it's working! If you're more experienced with Arduino and know how to test the input values of your particular sensors, you can adjust the range in the program so it works best for you. I assume all the sensors are practically identical,.If the servos aren't working properly, make sure all the connections are correct .

6. FINAL STEP : WIRING OF THE TOTAL CIRCUIT WITH EXECUTION

After completing all the steps mentioned above we assembled all the stationary circuits together to build tha main circuit for execution.The main circuit diagram is given below:-

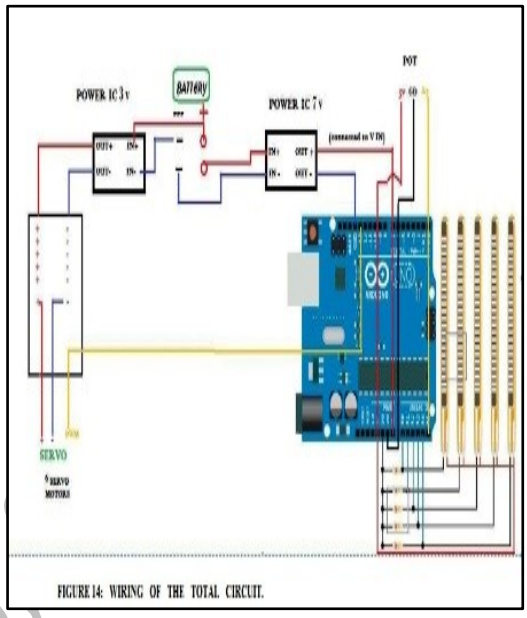


FIGURE 14: WIRING OF THE TOTAL CIRCUIT.

The final step is to upload the Arduino sketch to our board. As soon as the Arduino resets, the hand should start its calibration routine by moving each servo through its range of motion.

First after uploading the program to the Arduino and making sure all the connections to the glove and servos are correct we put on the glove and have plugged in the Arduino and servo battery and run the program. Putting the glove on and flex the finger that corresponded to the servo we were working on. Then we adjusted the servo ring. The servos rotated basing on how much fingers were bent. So the program started working.

C. MILESTONE CHART

Research Sensor Options	**Completed**
Research Servo Options	**Completed**
Order Parts As Needed	**Completed**
Setup ARDUINO	**Completed**
Configure Power IC	**Completed**

Interface	
Interface With Servo Motors	**Completed**
Interface With Flex Sensor	**Completed**
Construction Of Hand	**Completed**
Assembling The Hand	**Completed**
Mounting the Servos	**Completed**
Mounting the Flex Sensors	**Completed**
Calibrating The Flex Sensors	**Completed**
Write ARDUINO Sketch	**Completed**
Testing The Completed Code	**Completed**
Trouble-shooting	**Completed**

VI. TROUBLE SHOOTING

A. GENERAL SAFETY RULES :

1. As with any machine with moving parts, be careful not to let body parts get pinched. While the hand has very little gaps and it may be difficult to get trapped in them, it could potentially catch on clothing, rings, etc.

2. When operating this hand apply a maximum of 5 VDC to the micro servos. Anything above this voltage will destroy the small servos and will void the warranty.

3. When working on the hand make sure all power to the servos is turned off.

4. When the hand is at rest and not performing all power should be turned off.

5. While this hand has compliance (mechanical "give") built into all fingers, it is still possible to cause bodily and collateral damage with this hand if it's attached to a mechanism that is not carefully controlled.

6. If excessive force is applied to any finger or thumb for picking up heavy objects it may destroy the servos or can destroy any mechanism.

7.Our animatronic hand is equipped with SG 90 Micro Servos. At any point in this hand's lifespan:

- DO NOT Exceed 5 volts DC (VDC) to the servos. The power IC is fixed to be adjusted to a 3V max volt battery supply,which is ideal to ensure this voltage is never exceeded.
- DO NOT apply excessive force to the fingers / thumb. This is multiplied by the time it translates to torque on the servo horn. This will cause damage, typically resulting in "stripped" gears but can also damage the internal servo electronics.
- DO NOT get this hand wet. The mechanisms contain steel hardware which will rust if not cared for. The servos are not water-resistant either.

8.All IN+ and IN- pins of the Power IC have to be connected to the corresponding power supply pins.

B. COMPLIANCE ADJUSTMENT:

1. Each finger (including the thumb) is equipped with a spring.
These systems can be adjusted to compress springs more or less in each finger.

Less compression in the springs = more compliance in the finger

If the springs are not adjusted equally per finger this will shift the neutral position of the finger (which can be beneficial at times)

2.When programming the hand in whichever servo controller / microcontroller system you choose:

- Be very careful to define limits on each finger / thumb. The new hand has shipped with all servos at a "0" position (roughly translating to 1500 ms). At this position we have installed and tuned the finger / thumb mechanisms to be at a relative "0" position. This means that as you power up the hand for the first time and without inputting any position commands other than "0" or "neutral" the hand will move very close to the neautral position .

3.**To find the end limits for each finger / thumb**:
Because of the infinite position possibilities you have when adjust this hand's compliance, it is impossible for to predict – for each system – what those limits will be on the control system. You will have to set these yourself.

- To set the end limits first unplug ALL servos except the servo you are now tuning.
- Using this servos address on your control system carefully move the servo using whichever commands are applicable to the ARDUINO. Move this servo in small increments and LISTEN. When you hear the servo slightly "hum" and do not see any more motion in the finger / thumb in the direction you are traveling BACK

THE SERVO DOWN immediately. Leaving it at this point will damage the servo. By backing the servo down (changing the value to slightly closer to your neautral position value) you will extend the life of that servo and battery system powering the hand (stalled servos pull more amps than servos with a near-zero load).

4.Repeat the above for the opposite direction of that finger / thumb.

5.Repeat this process above for each finger.

C. RISKS AND INTERFACED ISSUES ENCOUNTERED WITH THE HARD WARES

1. BREAD BOARD BURNT OUT

Many small test circuits had to be built on breadboards and tested in the project, under actual circumstances, to make sure they would not fry.

- **CAUSE**: Several components were not grounded and did in fact fry. A breadboard was also melted.
- **TROUBLE SHOOT**: The circuit components must have to be grounded properly when needed.

2. FLEX SENSOR READING PROBLEMS

- **CAUSE:** This problem maybe caused due to the wrong design of our analog circuit.
- ✓ **TROUBLE SHOOT** :

1.We connected in series: a 5 Vdc supply, the flex sensor, 22K resistor and ground. (That is, the end of the resistor that is not connected to the sensor gets connected to the ground of the 5Vdc power supply.)

2.Then we connected a voltmeter between that same ground and the connection between the sensor and the resistor. Measured the voltage with various amounts of flexing. That should tell you what the analog input (AI) pin of your Arduino will see. Once we had that circuit giving us reasonable results, we went back for getting the Arduino to read that for us.

3. FINGERS WERE NOT MOVING PROPERLY

- **CAUSE**: While it was not moving, a noise from the servo was a steady "whirring" sound. May be we have applied too much force to the servo and stripped the internal gears.
- ✓ **TROUBLE SHOOT** :

1. We reset the Arduino as we were using one power source. Then Checked our power supply. Checked whether it was connected or not, Then checked whether it supplying at least 3.5 volts that minimum servo controller's recommended voltage or not.

2.Double checked our servo controller. Checked whether it was receiving / sending signals appropriately or not. Then we tried plugging a loose, new servo into the same port to test..

3.Inside the micro servo the motor is grinding plastic against plastic. After the proper testing we considered it as the main reason behind the problem we encountered . That's why we tried SG90 sevos of metallic gear instead.

4.Slight adjustments in the servo mounting holes may be necessary. This modification would void the warranty but may help your particular situation.

4. HAND/ WRIST FALLING OFF

- **CAUSE :** Lower clutch mechanism can't be sufficiently tightened to raise the arm.

- ✓ **TROUBLE SHOOT**

1. As this occurred at the beginning stage, I meant at the first try after assembly, we moved the corresponding motor and checked whether the gears were correctly engaged or not. As they were engaged properly we moved to the second trouble shooting process.

2. We checked whether the tapping screws were being installed on the wrist properly or not. If they were, then it has been possible the uppermost screw of the base side being not tightened enough. So, we tightened that appropriately and thus succeeded to trouble shoot.

5. SERVOS MOVING ERRATICALLY

- ❖ **CAUSES WITH TROUBLE SHOOTING:**

As with any animatronic project, many times this will come down to a complex system interaction between our microcontroller / pc and the servo controller. Usually the servos are just fine. Here are some things that have been found with testing done on multiple PC's and servo controllers:-

- ➢ **CAUSE-1** : Mechanisms may have been subject to dirt or adverse conditions. This can be confirmed by disconnecting the mechanism from the servo.

- ✓ **TROUBLE SHOOT**:
  We disconnected the mechanism from the servo then again connected the mechanism. If there were still issues the hand would have to be carefully disassembled, cleaned, re-assembled and tuned as above.
  But it didn't work either.

- ➢ **CAUSE-2:** The of cause of this can be almost always a "noisy" power supply. Even though we may had carefully selected an AC to DC 5VDC output supply that can handle 1 A pull, it may not be producing a quiet voltage.

- ✓ **TROUBLE SHOOT:**

  1.Any control system can be riddled with connection issues. Servos are usually the most trouble-free devices in the entire loop.,So. We backtracked into our servo controller settings (rate mis-match, etc), cable connections, ARDUINO interference, and our code.

  2.As the above did not solve the issue,we tried plugging in a simple battery supply of about 11.1 VDC and then converted the voltage of the supply into 3VDC which is the threshold voltage of the servo by adjusting the voltage of a power IC. In all cases this has corrected this issue in our testing.

- ➢ **CAUSE-3**:. If your powering the servos with a different power source than the Arduino, they need to share the same ground or the PWM signals will corrupt between the Arduino and servos. Fingers / thumb are moving but erratically.

- ✓ **TROUBLE SHOOT**: We checked all our ground connections properly as the above trouble shooting process couldn't resist the servos form moving erratically in total . After that we reconnected those thus we managed to solve this problem finally.

6. SERVOS MOVING SLOWLY WHEN IT SHOULD BE STOPPED

- ➢ CAUSE: This was a result of our servos not being calibrated quite right.

- ✓ **TROUBLE SHOOT**: To calibrate our servos we had to wait until our program had the servos stopped. Then with a small screwdriver we tuned the potentiometer inside the servo (the little opening in the servo casing just above where the wires are). Then rotated that either left or right until the servo was truly stopped.

7. SERVOS BEING SO NOISY

- ❖ **CAUSES WITH TROUBLE SHOOTING :**

The word servo refers solely to a device that uses negative feedback for control. One major drawback to working with servos is the large amounts of electrical noise they produce.This noise can interfere with your sensors and can even impair your microcontroller by causing voltage dips on your regulated power line. Large enough voltage dips can corrupt the data in microcontroller registers or cause the microcontroller to reset.

- ➢ **CAUSE-1:** cheap brushed motors can be noisy. Cheap hobby grade servos can sometimes chatter if they do not settle in a stable state. This is normal and is caused by poor tuning, a lack of a dead band, and backlash between the motor and the encoder (potentiometer).

- ✓ **TROUBLE SHOOT**:

  1.We can get very quiet systems if you are willing to pay for it.

  2. Keeping our motor and power leads as short as possible, we can decrease noise by twisting the motor leads so they spiral around each other.

- **CAUSE-2:** The main source of motor noise is the commutator brushes, which can bounce as the motor shaft rotates. This bouncing, when coupled with the inductance of the motor coils and motor leads, can lead to a lot of noise on your power line and can even induce noise in nearby lines.

  - ✓ **TROUBLE SHOOT** : We didn't go for the trouble shooting of this as we found the circuit to be more complicated to be executed properly. That's why we bought the new SG90 motors with metalic gears and thus tried to troubleshoot this problem.

    The further discussion about this Trouble shooting process will be written on the **ASSUMPTIONS AND LIMITATIONS** portion of this report**.**

D. INTERFACED ISSUES ENCOUNTERED WITH THE SOFTWARE

1.COMMON CODING ERRORS:

- Forgetting the semi-colon at the end of a statement
- Misspelling a command
- Omitting opening or closing braces
- Thus If there is a syntax error in the program caused by a mistake in typing, an error message will appear in the bottom of the program window.

✓ **THOUBLE SHOOTING:**

Generally, staring at the error will reveal the problem. If we continue to have problems,
we should try these ideas:

- ✓ Running the Arduino program again.
- ✓ Checking that the USB cable is secure at both ends.
- ✓ Rebooting our PC because sometimes the serial port can lock up. If a "Serial port…already in use" error appears when uploading.

2. ERRORS ABOUT UNDECLARED FUNCTIONS OR UNDECLARED TYPES:

- **CAUSE :** The Arduino environment attempts to automatically generate prototypes for our functions, so that we can order them as we like in our sketch. This process, however, isn't perfect, and sometimes leads to obscure error messages.

  If we declare a custom type in our code and create a function that accepts or returns a value of that type, we'll get an error when we try to compile the sketch. This is because the automatically-generated prototype for that function will appear above the type definition.

✓ **TROUBLE SHOOT**: If we declared a function with a two-word return type (e.g. "unsigned int") the environment would not realize it's a function and would not create a prototype for it. That meant we needed to provide our own, or place the definition of the function above any calls to it.

VII. ASSUMPTIONS AND LIMITATIONS

A. USING OF FLEX SENSORS IS NOT COST EFFECTIVE

LIMITATION :

Flex sensors are so much costly. It took a huge amount of money to be spent on the implementation of this process. It's one of the main limitations of this project.

We could have used some other process to make the sensors by ourselves, But thinking of the time limit of the academic submission of this project and also for the perfection of the execution of this project we used the costly Flex sensors.

ASSUMPTION:

We could have used the Neopren Bend Sensors in place of the flex sensors for the movement of the fingers of the animatronic hand.
It would be of so much cost-effective but time-consuming.

This bend sensor actually reacts (decreases in resistance) to pressure, not specifically to bend. But because it is sandwiched

between two layers of neoprene (rather sturdy fabric), pressure is exerted while bending, thus allowing one to measure bend (angle) via pressure. It is sensitive enough to register even slight bend and has a large enough range to still get information when the limbs are fully bent. The neoprene is great for isolating the conductive thread stitches and keeps the sensor from wrinkling even when repeatedly being bend.

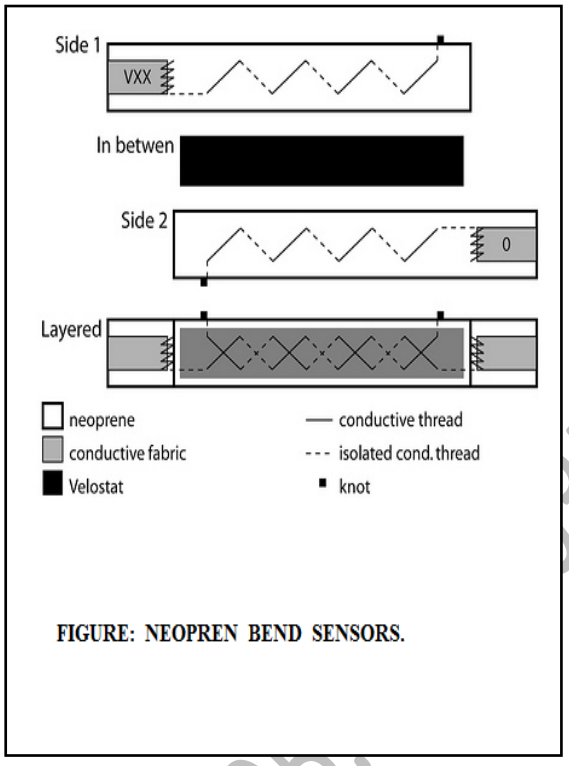


FIGURE: NEOPREN BEND SENSORS.

The resistance range of this bend sensor depends a lot on the initial pressure. Ideally you have above 2M ohm resistance between both contacts when the sensor is lying flat and unattached. But this can vary, depending on how the sensor is sewn and how big the overlap of the adjacent conductive surfaces are. Sewing the contacts can be done as diagonal stitches of conductive thread – to minimize the overlap of conductive surface. But only the slightest bend or touch of the finger will generally bring the resistance down to a few Kilo ohm and, when fully pressured, it goes down to about 200 ohm. The sensor still detects a difference, right down to about as hard as you can press with your fingers. The range is non-linear and gets smaller as the resistance decreases.

B. USING OF UNNECESSARY FLEX SENSORS

LIMITATION :

We used five flex sensors for the movement of the five fingers of our animatronic hand.

But due to some unavoidable circumstances one of the flex sensors has losen it's sensitivity. For the reason we needed to buy one more flex sensors. But as the flex sesnsors are costly, we switched to the another plan. We decided to connect the last two fingers with the animatronic hand with the same flex sensor. Thus the fourth flex sensor has been used to move the two fingers at a time.

ASSUMPTION: As the fourth flex sensor has been used to move the two fingers at a time. The other flex sensors could also be used for the movement of the finger (more than one) at a time. So, there were no need of buying exactly the five flex sensor for the movement of five fingers at a time. We assumed it unnecessary.

C. COULDN'T DEAL WITH THE MOTOR NOISE

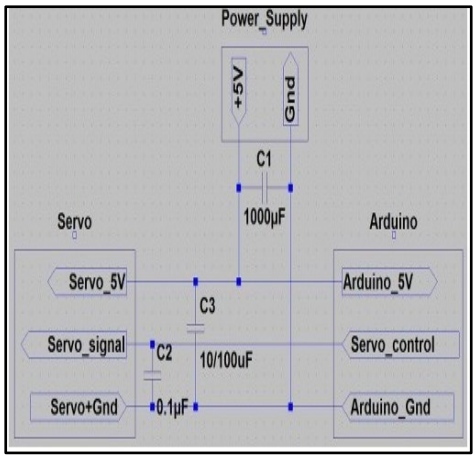
LIMITATION : As we found the trouble shooting circuit of motor noise a complex one and also due to the time limit of the project submission we couldn't deal with the motor noise. And we bought new motors to solve this problem.

ASSUMPTION: For dealing with the motor noise we assumed some of the measures which we could have executed in case of having some more duration of the project time-limit.

- **SOLDERING CAPACITORS ACROSS OUR MOTOR TERMINALS.**

Capacitors are usually the most effective way to suppress motor noise, and as such we recommend **always** solder at least one capacitor across motor terminals. Typically we will want to use anywhere 0.1uF capacitors, soldered as close to the motor casing as possible. For applications that require bidirectional motor control, it is very important that we do not use polarized capacitors!

- We can construct the following circuit if the above measures cannot remove the motor noise.



VIII. CONCLUSION:

We are glad that we chose to complete this project on the Arduino. It was our first real coding experience on this platform, and we can say that compared to writing C++, writing Wiring libraries for Arduino makes for a much more fun and Productive experience We are grateful that our time on the C++ taught us a lot about what is happening behind the scenes, but quiet honestly it is nice to not have to worry about it so much.

One thing we learned from this project is that servos and flex sensors in positioning, timing and environmental texture can lead to all sorts of undesirable readings. We were a bit disappointed with the performance of the SG90 servos in this particular use case, It required a lot of the fine-tuning to get readings accurate as the servo rotated.

As stated previously, another area I need to look into is battery power. This project is a poor use case for a 11.1V battery. A better long term portable power supply would include a higher efficiency in the output. That's why we also used a PC for the long term power supply.

Although the Animatronic hand did not operate with no errors, it is a great success overall. The Animatronic hand met all safety restrictions, easy to operate and energy efficient. This types of animatronic hand can be used for various puposes. The Animatronic Hand can be implemented in all the sectors where human interaction is needed, like-Handling of the explosive objects, performing various sophisticated operational jobs in the medical sectors, Industrial manufacturing etc.

With more time and resources put for things like motors and base design we can carry a much larger payload and have a sturdier platform to carry things in. Much of this project could be used or improved upon by future EEE students.

IX. LESSON LEARNED

The risks were subject to like everyone else, running the risk of not receiving all materials in time. Also, the calibration of flex sensors and the servos proved to be difficult. We also had to be careful about the ARDUINO and its wiring. We also needed to make diagrams so we did not detach a bunch of wires and not know how to rewire them. Due to the recent political affairs and with the improper communication facilities,we also ran the risk of not having enough man-power to complete the project. This is why we needed to start as early as possible. We had to work very hard to complete this project in time! But in the end, the challenge and learning experience were well worth it.

X. TASKING AND SCHEDULING

1. Gathering the hard wares ⟶ 1 week.
2. Construction of the hand and making the glove ready ⟶ 1 week.
3. Calibration of flex sensors and testing of servos ⟶ 1 week.
4. Arduino wiring and Arduino code ⟶ 2 weeks.
5. Connecting the whole circuit with trouble-shooting ⟶ 1 week.

XI. BILL OF MATERIALS

COMPONENTS	PRICE
1.ARDUINO MEGA2560	1*2,000= 2,000T.K
2. SERVO SG90	5*500 =2,500T.K
3. SERVO 14K TORQUE	1*1,200=1,200T.K
4.SERVO 10K TORQUE	2*1000=2,000T.K
5. BREAD BOARD	1*180 =180T.K
5. JUMPER WIRE	3*150 = 450 T.K
6. BATTERY	1*1,800=1,800T.K
7. CHARGER	1*1,500=1,500 T.K
8. FLEX SENSOR	5*2,100=10,500T.K
9. BOSTER	2*250 =500T.K
10.POT	2*10 =20T.K

11.RESISTORS	5*2 =10 T.K
12.SPRING	10*5 =50 T.K
13.CHARGER CONNECTING CABLE	1*160=160 T.K
14.PCB	1*20=20 T.K
15.WOOD & PLASTIC PIPES	100 T.K

TOTAL=22,990 T.K

XII. REFERENCE

[1] **The Absolute Beginner's Guide to Arduino**

http://forefront.io/a/beginners-guide-to-arduino

[2] **Principles of Robotics**
http://www.g9toengineering.com/resources/robotics.htm

[3] Servo Motor | Servo Mechanism | Theory and Working Principle
http://www.electrical4u.com/servo-motor-servo-mechanism-theory-and-working-principle/

[4] DC Servo Motors | Theory of DC Servo Motor

http://www.electrical4u.com/dc-servo-motors-theory-and-working-principle/

[5] Introduction to Servo Motors

http://mechatronics.mech.northwestern.edu/design_ref/actuators/servo_motor_intro.html

[6] DIY Robotic Hand Controlled by a Glove and Arduino

by dschurman
http://www.instructables.com/id/DIY-Robotic-Hand-Controlled-by-a-Glove-and-Arduino/

[7] Animatronics

http://en.wikipedia.org/wiki/Animatronics

[8] Simple Servo Bed for InMoov

http://www.thingiverse.com/thing:65274

[9] AniHand - Animatronic Hand

http://letsmakerobots.com/node/34639

[10] Calibrator: An Arduino library to calibrate sensors hooked to analog inputs

http://julianvidal.com/blog/calibrator-an-arduino-library-to-calibrate-sensors-hooked-to-analog-inputs/

[11] Arduino - Begin.

http://arduino.cc/en/Serial/begin

[12] Arduino learning pdf

http://www.ele.uri.edu/courses/ele205/Arduino%20-%20Learning.pdf

[13] Arduino Microcontroller

Guide
http://www.me.umn.edu/courses/me2011/arduino/arduinoGuide.pdf

[14] Arduino functions
http://playground.arduino.cc/Code/Function

[15] Structuring Your Code into Functional Blocks

https://www.inkling.com/read/arduino-cookbook-michael-margolis-2nd/chapter-2/recipe-2-10

[16] Arduino programming Problem solution

http://stackoverflow.com/questions/12975895/arduino-argument-of-type-void-classname-does-not-match-void-whi

[17] Blink Without Delay
http://arduino.cc/en/Tutorial/BlinkWithoutDelay

[18] Operating Two Servos with the Arduino

http://www.robotoid.com/appnotes/arduino-operating-two-servos.html

[19] Servo Problems With Arduino

http://rcarduino.blogspot.com/2012/04/servo-problems-with-arduino-part-1.html

[20] Problem with arduino servo

http://electronics.stackexchange.com/questions/93746/problem-with-arduino-servo

[21] Arduino Servo control problem, external servo power source
https://forum.sparkfun.com/viewtopic.php?f=32&t=24263

[22] Problem with multi servos on Arduino

http://www.youtube.com/watch?v=iZFO_8hH7QY

[23]Multiple servo control

http://www.instructables.com/id/Serial-Servo-Controller-wAduino-Control-Up-To-1/

[24]Flex sensor

http://mech207.engr.scu.edu/SensorPresentations/Jan%20-%20Flex%20Sensor%20Combined.pdf

[25] Arduino map() method

http://stackoverflow.com/questions/9024124/arduino-map-method-why

[26] Arduino programming tutorial

http://opensourcehardwaregroup.com/tutorial-09-reading-analog-pins-and-converting-the-input-to-a-voltage/

[27] Calibration of flex sensors

http://www.mtbs3d.com/phpBB/viewtopic.php?f=138&t=17799

[28] Sensing A Bend With A Flex Sensor + Arduino

http://bildr.org/2012/11/flex-sensor-arduino/

[29] Arduino tutorial PWM

http://www.youtube.com/watch?v=Y1QraI5i_XM