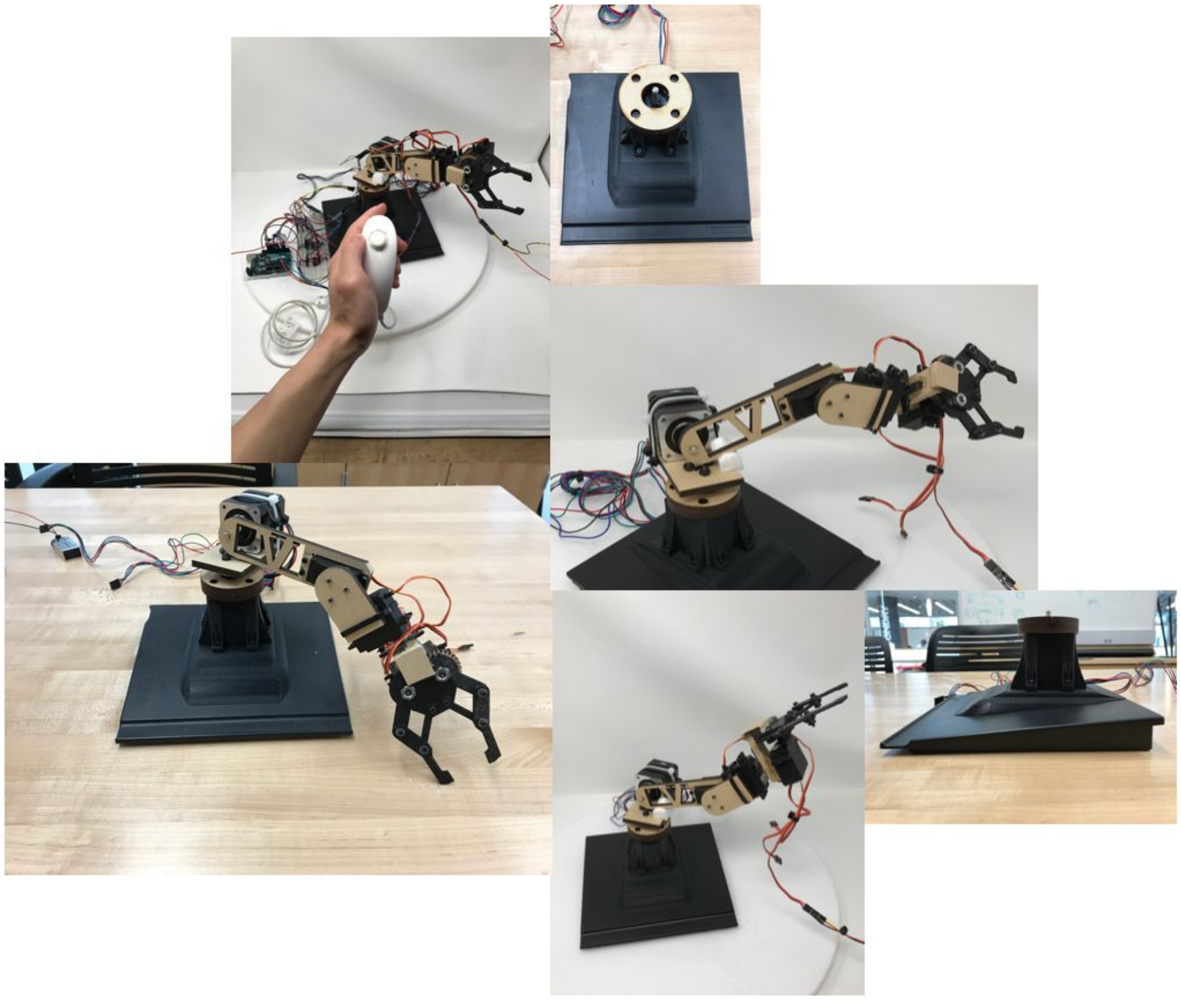
Nintendo Wii ® Nunchuck Controlled 5 DOF Robotic Arm

Dennis Sohn[1]

[1] *Vanderbilt University School of Engineering, Mechanical Engineering*



ME 3204 Mechatronics Final Report

Professors: Michael Goldfarb, Ph.D

Kenneth Frampton, Ph.D

I. Introduction

Motivation

Robotic manipulators have become more and more prevalent in various sectors of society in the past three decades. One particular field where robotics has grown most prevalently is Medical Robotics. More and more health professionals have begun transitioning from utilizing manual medical devices to taking advantage of the more advanced and precise control medical robots that have been developed. As an aspiring engineer who has had the opportunity to participate and contribute in research concerning medical robotics in STORM Lab at Vanderbilt, I was inspired to create a robotic manipulator that could have medical applications. Intuitive Surgical ®, a well established and respected company in the field of medical robotics, have developed a four arm surgical system named the DaVinci Surgical System ® (**Fig. 1)**. The DaVinci has allowed for surgeons to operate on patients with higher precision and in a minimally invasive manner, allowing for smaller cuts, less bleeding, and quicker recovery time. But along with innovation various projects in the biomedical engineering arena focus on creating effective, but also low cost application devices in order to help societies of low economic development with high health risks and demands. My general motivation behind medical robotics is to help develop systems that can save countless lives for those who would otherwise not be able to afford the medical care that is currently available.

Figure 1: *Intuitive Surgical's DaVinci Surgical System ® Utilizes a Four Arm System Controlled by a Medical Doctor to Perform Operation*

General Function

The WiiArm is a five degrees of freedom (DOF) arm which is controlled by a commercial sensor controller called the Nintendo Wii Nunchuck. The user controls the arm by manipulating the nunchuck's different input sensors and controllers. By doing so, the user can control 5 end effector behaviors: 1) Rotating the entire arm in the xy-plane 2) Actuating the entire arm in the vertical plane 3) Actuating a section of the arm in the vertical plane 4) Rotation of the end effector claw mechanism in the axial direction of the arm 5) Actuation controlling the opening and closing of the end effector claw mechanism.

The nunchuck outputs three main sets of data, which are used to manipulate the arm: 1) Accelerometer data in the $\mathbb{R}^3$ Cartesian Plane 2) A joystick with data points in XY-Cartesian coordinate plane 3) "C" and "Z" button outputting a binary On/Off response signal

The accelerometer and joystick coordinate points were used to control the translational and rotational motions of the arm while the "c" and "z" buttons controlled the opening an closing of the claw mechanism. This gave the user, who was operating the nunchuck, full control of the WiiArm.

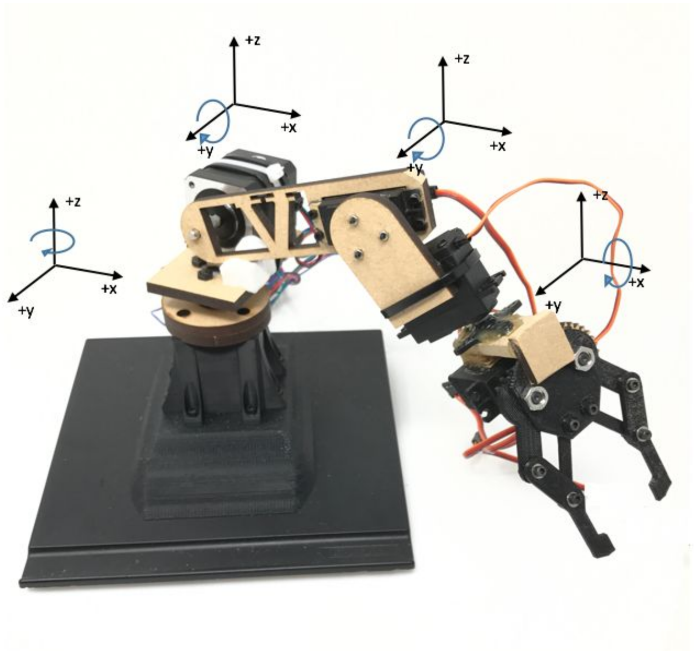


Figure 2: *Four Axis of Motor Rotation for the WiiArm*

II. *System Description*

Design: Hardware & Construction

A base support held the entire system, which was mounted onto it using 8 M2 5 x 14 screws and a base support piece and the stepper motor encasing. Both parts were rapid prototyped and created using Autodesk Inventor Professional 2017 ® and fabricated by the Stratsys 3D Printer.

The NEMA 17 Stepper Motor, which is enclosed by the encasing piece was oriented such that the motor shaft was pointed in the +Z axis. Two plates 60mm in diameter with extruded holes positioned in the same position as the threaded holes on the face of the stepper motor. The plates were laser cut from MDF wood and then attached to the stepper. A third plate was laser cut with a hole in the center that had the projected geometry of the stepper motor shaft. The stepper motor's shaft is 4.5mm in diameter with a cross-diagonal cut.

A second stepper motor was positioned normal to the axis of the first stepper motor such that the motor shaft was parallel to the xy-coordinate plane. This stepper motor was offset from the center axis of the first stepper motor by 11.5 mm such that the second stepper motor's shaft was centered to the first stepper motor.

The robotic arm consisted of two main joints limbs: a major arm and a minor arm. The major arm (***Fig. 2***) included a hole on the proximal end with the stepper motor shaft cross sectional projected geometry and a rectangular slot of dimensions (insert) for a (insert) servo motor on the distal end. The servo motor was held in place using (insert) screws.

The minor arm (***Fig. 2***) was attached to the major arm by way of the servo motor shaft and attachment rotor piece. Due to limitations in maximum allowed torque, the minor arm is much shorter than the major arm to reduce the perpendicular distance and force and thus resultant torque acting on the servo motor shaft. At the distal end of the minor arm, a second servo motor is attached with the motor shaft in the direction of the y-axis of the minor arm.

At the distal tip of the arm is a four-link claw grasping mechanism, which is actuated by a third servo motor. The claw was designed and prototyped using Autodesk Inventor and fabricated by the Stratsys® 3D Printer. The claw was assembled using an array of eight (specify) screws, which were positioned at the terminal ends of each the joints inside cylindrical clips, allowing for rotation.

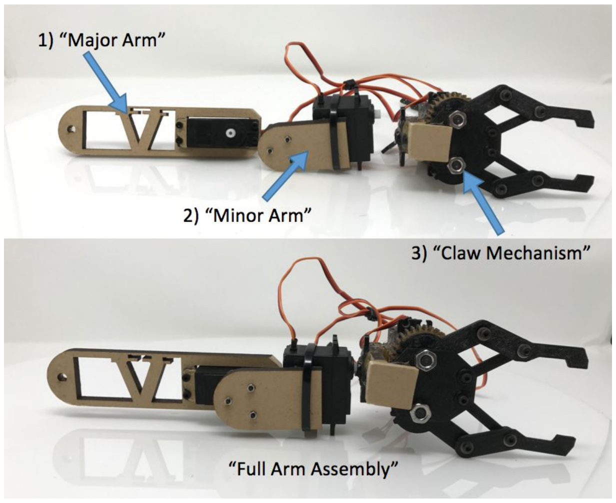


Figure 2: *WiiArm Components Assembly*

Actuation Units

Two types of actuation units were used in the WiiArm. The first was a NEMA 17 Bipolar 40mm, 2 Amp, 4 Lead Stepper Motor (**Fig. 3**). Stepper motors operate by drawing current for each phase, which allows the motor to bear a greater load or torque compared to traditional DC Motors. Current control The NEMA 17 Stepper motor was rated to draw 2A of current and take on a holding torque of 45N-cm. The stepper motor moves by rotating in steps with a designated step angle. The NEMA 17 Stepper Motor has a Step Angle of 1.8 degrees, which translates into 200 steps for one full revolution ($1.8^o \times 200$ Steps $= 360^o$).

The Stepper motor was specifically chosen to actuate the base of the arm because of its high torque capabilities. The first stepper motor took on a high load in the axial direction of the shaft as the entire arm rested on it. The second stepper motor took on a high load in perpendicular direction (torque) as a resultant of the ascending and descending motion of the arm. A weaker motor would not have had enough torque to withstand the forces acting on it and would have limited the WiiArm's function movement.

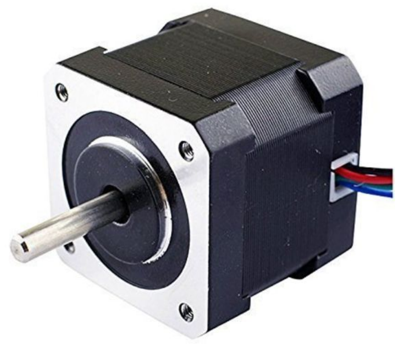


Figure 3: *NEMA Stepper Motor*

The second actuation unit was the Fitec® FS5103B Standard Servo Motor. This servo motor operates based on pulse width signals given from the arduino. Servo Motor Standard operates between a pulse width of 900 ms and 2100 ms where 900 ms scales to a position of -70^o and 2100 ms scales to a position of $+70^o$. 1500 ms centers the position of the servo motor to 0^o. These motors are effective in lower torque actuation situations where input data from a sensor such can be converted and mapped to digital output data scaled for the servo motor position. The Servo Motor Standard was rated for 6V and could handle a torque of 2.5 kg-cm.

Figure 4: *Fitec® FS5103B Servo Motor*

The digital input data from the Wii nunchuck, the accelerometer, joystick, and “c”/ “z” button values were scaled and mapped to match with the range and positioning of the servomotors. This allowed for rotational motion of the smaller components of the arm at a lower overall effective weight for a motor, which was important when considering how the forces of weight would affect the torque of the stepper motors at the base.

Circuitry

The WiiArm was controlled using an Arduino Uno® R3 Microcontroller and Arduino software (**Appendix**). An 830 Point Solderless Breadboard (**Appendix.)** was used to construct the circuitry connecting the actuation units of the arm and the Arduino. Electrical wire connectors were used to connect all components.

The Stepper Motors were controlled using an electrical element called the H-Bridge (**Fig. 6**). The H-Bridge used for the WiiArm was a Texas Instrument® L293NE Quadruple Half-H Driver (**Fig. 5**). The H-Bridge allowed for bipolar or bidirectional control of the stepper motors using a series of voltages gates set to “high” or “low”, which controls current flow and thus dictates either a clockwise, counterclockwise, or neutral step response from the motor. Four pins of the H-Bridge connected to the Arduino digital output pin. In correspondence, four output pins connected from the H-Bridge circuit connected to the Stepper Motor.

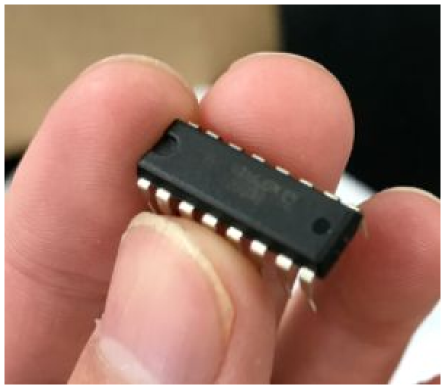


Figure 5: *TI L293NE H-Bridge*

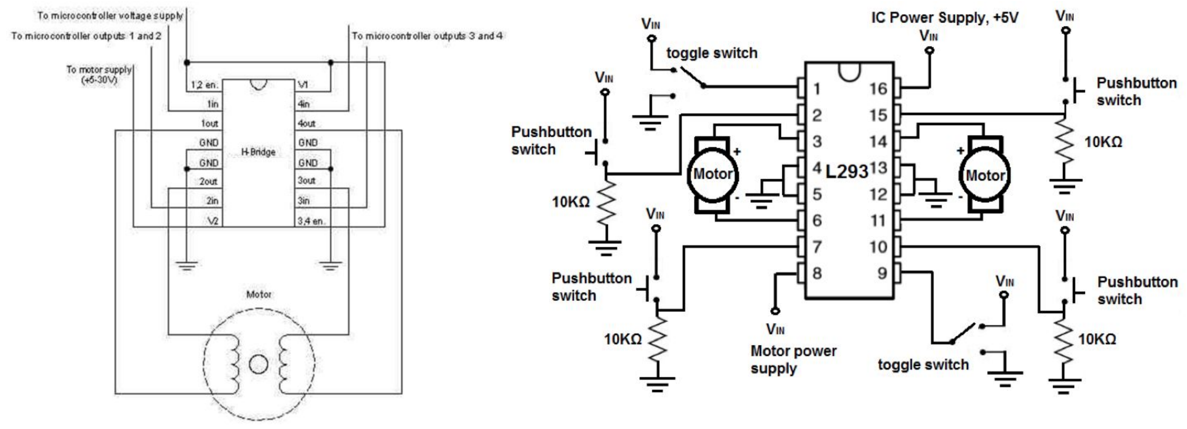


Figure 6: *Circuit Schematic of TI L293NE H-Bridge*

Due to the stepper motor being rated at a high voltage and current, a power supply source machine was used to power the stepper motors instead of the Arduino or 9V battery. The power source allowed for voltage and current control in order to provide the necessary voltage at a controlled current to prevent overheating of the H-Bridge.

The servo motors were rated at a 6V max V_{in} for full power. Due to the high torques acting on the motors, the servo's max voltage needed to be taken advantage of. The Arduino only supplied 5V, so a 9V battery and voltage divider was created to supply 6V to the servo motors. In order to acquire 6V from 9V, three 220 kΩ resistors were put in series to satisfy the following Nodal Analysis equation:

$$\frac{9V-6V}{R1} = \frac{6V-0V}{R2}$$

$$\frac{R2}{R1} = \frac{220k\Omega + 220k\Omega}{220k\Omega} = 2$$

Each servo motor also connects to Ground and a voltage input which is connected to the Arduino. The Arduino outputs voltage signals to control pulsewidth, which in turn controls the servo's rotor position.

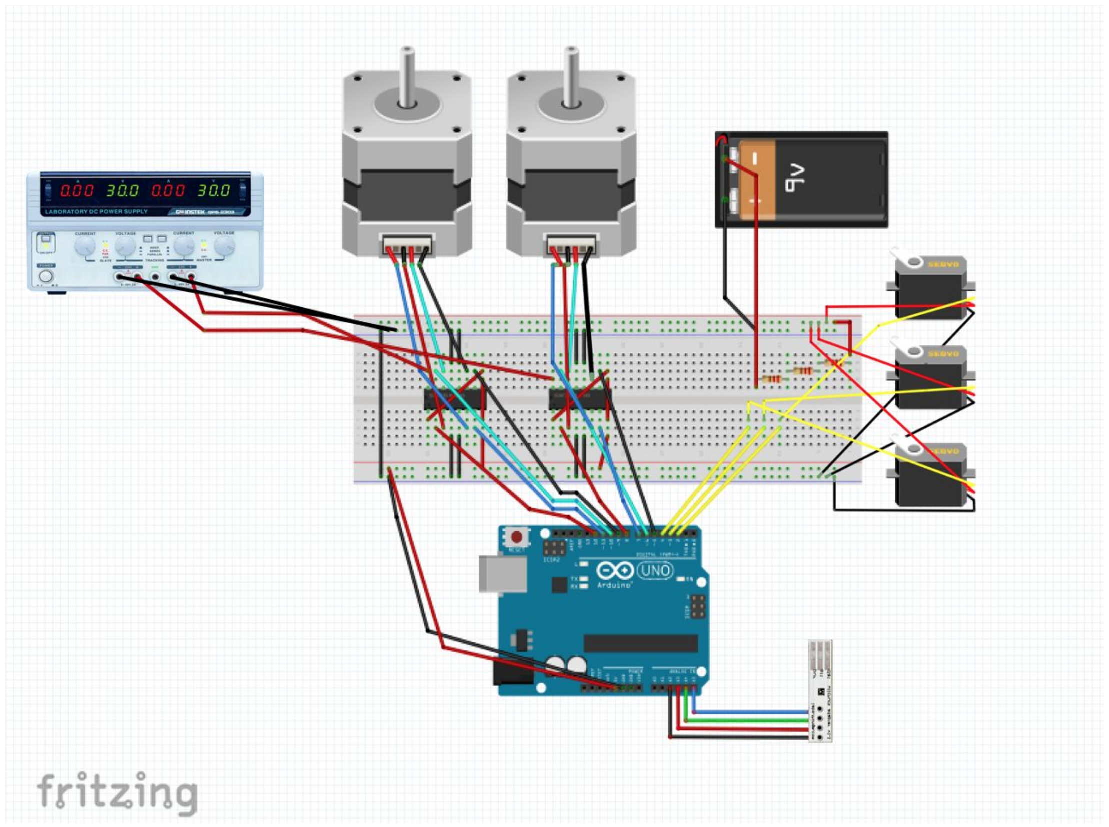
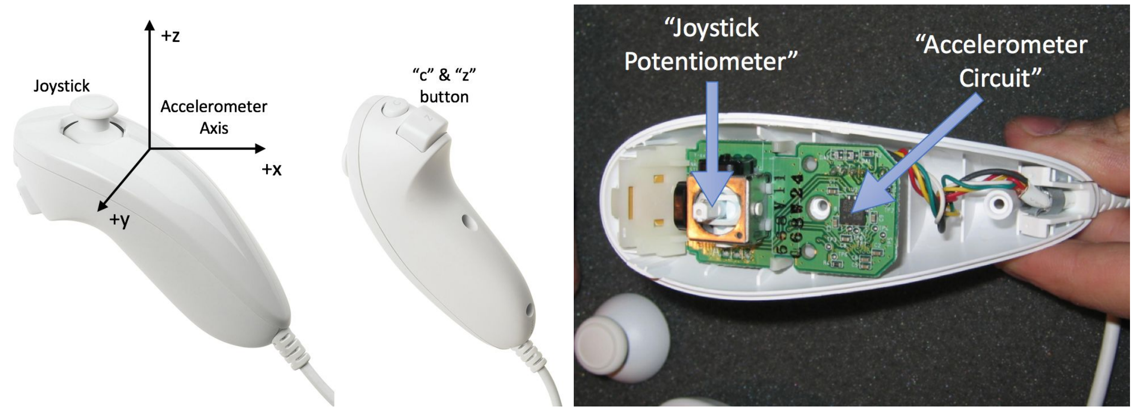


Figure 7: *Overall Visual Schematic of the Circuit Control of 2 Stepper Motors, 3 Servo Motors, Power Supply, 9V battery, WiiChuck Adapter, and Arduino Uno Microcontroller (created with Fritzing® Software)*

Sensors

The sensor used to control the entire arm was the Nintendo Wii ® nunchuck (**Fig. 8a,b**). The nunchuck consists of: 1) Accelerometer, which outputs gravitational force data in the x, y, and z directions 2) Joystick, which is constructed with a curvilinear potentiometer that varies resistance based on the position and outputs data in the x and y direction 3) Two pushbuttons ("c" & "z) which output a binary high (1) or low (0) depending on whether the button has been pushed or not.



(a) *(b)*

Figure 8a, b: *Anatomy of Wii Nunchuck: Accelerometer Circuit, Joystick, Push Buttons*

Through simulation and gathering of data, the maximum and minimum output value ranges were found for the accelerometer and joystick potentiometer. The Arduino was able to acquire the output data from the nunchuck using an adapter circuit called the wiichuck adapter (**Fig 9**). The wiichuck adapter consists of four pins, two for power (positive and negative poles) and two for data output (data and clock). The Wii Nunchucks encrypts data using a protocol called I^2C (Inter-Integrated Circuit).

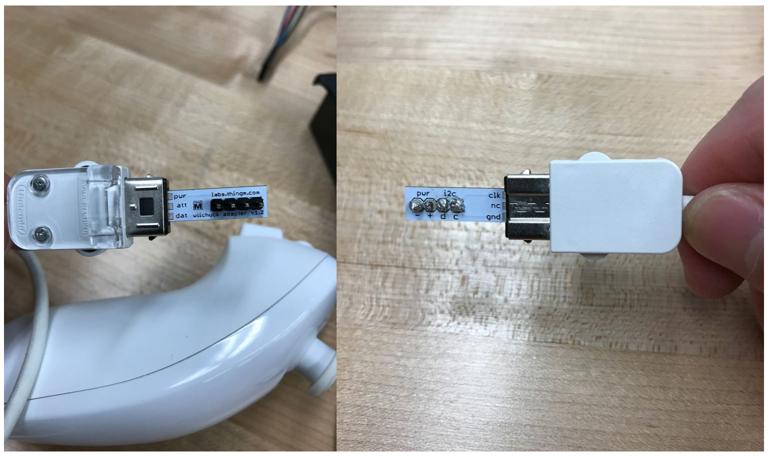


Figure 9: *WiiChuck Adapter Plugged into Wii Nunchuck*

Arduino Code Logic

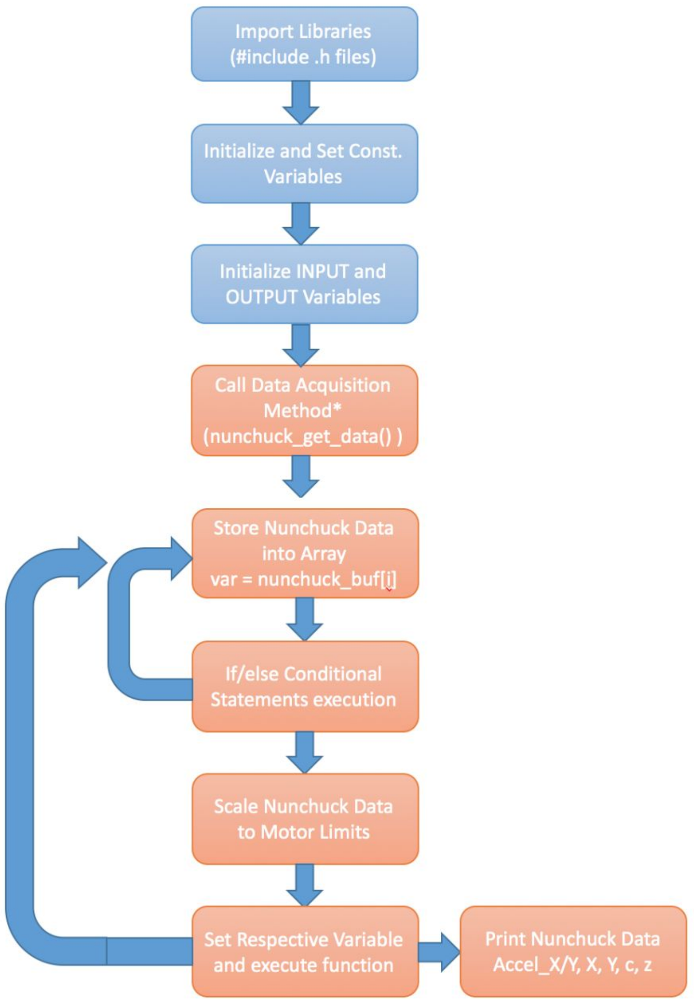

Import Libraries (#include .h files)
Initialize and Set Const. Variables
Initialize INPUT and OUTPUT Variables
Call Data Acquisition Method* (nunchuck_get_data())
Store Nunchuck Data into Array var = nunchuck_buf[i]
If/else Conditional Statements execution
Scale Nunchuck Data to Motor Limits
Set Respective Variable and execute function
Print Nunchuck Data Accel_X/Y, X, Y, c, z

Figure 10: *Flow Diagram of Arduino Code Used to Control WiiArm*

The Arduino software, which is rooted in the C-Language, is divided into two sections: a Setup and a Loop. The setup (represented in blue in the flow diagram) is run once and that is where variables are declared and initialized. The Loop (represented in red) runs continuously over and over again until the program is stopped. This shown in **Fig. 10**.

Two header files from the Arduino library were imported, the wire.h file and stepper.h file. These libraries contain predefined functions for the wire transmission functions used in acquiring data from the nunchuck and stepper motor command functions when controlling stepper motor output signals.

Variables used in the code were then declared and initialized for use in the program. This also included variables that would serve as Input and Output control variables. Once the initial variables were declared, the method nunchuck_get_data() was called to acquire the nunchuck's output data. nunchuck_get_data() used a programming technique called the I2C protocol which communicates with the data and clock pins from the nunchuck and converts it into readable digital signals. The data from the I2C protocol was stored into an array called nunchuck_buf[ ].

With the stored numerical values of the nunchuck output data into an array, the variables were then called in a series of "if/else" conditional statements to execute functions for the WiiArm's movement. If the condition is satisfied, then the variable's value will get passed into the conditional statement where a function is executed such as actuating the stepper or servo motor to perform a task. In this manner, the input signals coming from the nunchuck control the output behavior of the mechanical arm. If the condition is not satisfied, no action occurs and the code loops back to read in data until the condition is satisfied. The data is also printed to a "Serial Monitor", where requested variables were printed and displayed such as the values for the accelerometers, joystick, c and z button, and which motors were turning. This processed looped until the program was stopped.

III. Conclusion

Results

The WiiArm performed basic functionalities successfully, although not always consistently and in a synchronous manner. Each actuation unit was tested individually before the final assembly. Each stepper motor performed proper step rotation with the input signals of "joystick right" (corresponding to a clockwise rotation) and "joystick left" (corresponding to a counterclockwise rotation) as well as "joystick up" (corresponding to clockwise rotation causing the entire arm to elevate) and "joystick down" (corresponding to a counterclockwise rotation causing the entire arm to descend).

Each individual servo motor was also tested for functionality. The first servo controlling the ascending and descending behavior of the minor arm and claw, experienced the highest torque of the three servo motors as it took on the load of the other two servo motors as well as the claw mechanism. The second servo controlling the rotational orientation of the claw mechanism also performed with full functionality. The third servo motor successfully actuated the gear system to open and close the four-link claw mechanism.

When all of the components were put together, the response of the arm from the input of the data given from the Wii nunchuck, was unstable and slow at various times. It was noticed that the stepper motor's response was delayed by almost a full 2 seconds and the servo motor response varied from either immediate response to anywhere between 1-2 second delay as well.

It was also observed that the stepper motor was very limited in functioning when the load of the arm was applied as the H-Bridge max current capacity was only 0.6 A while the stepper required about 2 A. Thus, the stepper motors were not able to intake the current needed to successfully rotate the base of the arm. The servo motor systems, although inconsistent at times in response, overall functioned very well, performing their respective actuation functions. The minor arm ascended and descended with the ascension and descension of the nunchuck, as well as rotating the orientation of the tip with lateral rotation of the nunchuck, and most excitingly, the claw grasping mechanism successfully worked, opening when the user pushed the "z" button and closing when the user pushed the "c" button.

Summary

The Wii Nunchuck controlled robotic arm, overall, showed innovative potential. A simple controller such as the popular video game console had the capability to control a very sophisticated mechanical device with good precision. Range of motion and functionality was successfully achieved, although not on a consistent basis.

In the future, two main components for improvement manifest from this prototype. The first is replacing the current L293NE H-Bridge with a higher toleranced max current H-Bridge such as the ROB-12589 Big Easy Driver, which will allow for more current to be drawn by the stepper motors and allow full functionality in rotating, lifting, and declining the arm. The only main drawback to using this new H-Bridge driver is cost as the Big Easy Driver costs $19.95 per driver compared to $3.90 for the L293NE. However, for the sake of an optimally performing arm, a stronger driver is necessary.

The second change would involve the coding and microcontrol process of the system. One of the problems encountered during the operation of the WiiArm was the inconsistent response and behavior of the arm when processing the input signal data of the Wii Nunchuck. The Wii nunchuck was continuously outputting 7 data points (3 from the accelerometer, 2 from the joystick potentiometer, and 2 from the buttons). Along with the 7 inputs, 6 response controls were also continuously being outputted at the same by the Arduino. Because of the large amounts of data being processed continuously, timing and processing issues may have arose causing delayed responses. This problem may have been fixed by manipulating data storage in a more efficient way through C-Language data architecture manipulation techniques or using a stronger processing microcontroller.

Through constructing the first prototype of the WiiArm, the potential for further application and enhancement was validated and with enough refinement, the future of low cost robotic control could progressively move more and more towards video game-like control.

IV. Appendix A: Component Source

Part Name	Price
Arduino Uno (acquired from Kit)	$21.99
Base Mount (found in recycling bin)	Free
Mount Adjustment Piece And Motor Encasing (3D printed)	Free
Nema 17 Stepper Motor (Amazon.com)	$10.99/each
830 Point Solderless Breadboard (acquired from friends)	$5.95
Jumper Wires (adafruit.com)	$3.95 per 40 wires
Servo Motor Standard (acquired from kit and friends)	$8.50
Screws, Nuts, and Washers (acquired from Wond'ry)	Free
WiiChuck Adapter (acquired from STORM Lab)	$1.95
Nintendo Wii ® Nunchuck (acquired from home)	$6.50
Laser Cut Arms and Plates (Laser Cut in Olin)	Free
Zip Ties (acquired from Wond'ry)	Free
Resistors (acquired from kit)	Free
9Volt Battery (acquired from kit)	$5.99

Loctite® Glue and Hot Glue *(acquired from Wond'ry)*	***Free***
GW Instek 2303 Powersupply *(acquired from STORM Lab)*	***$313.00***
Total:	***$389.81*** ***$15.89****

***excludes items acquired for free**

V. Appendix B: Arduino Code

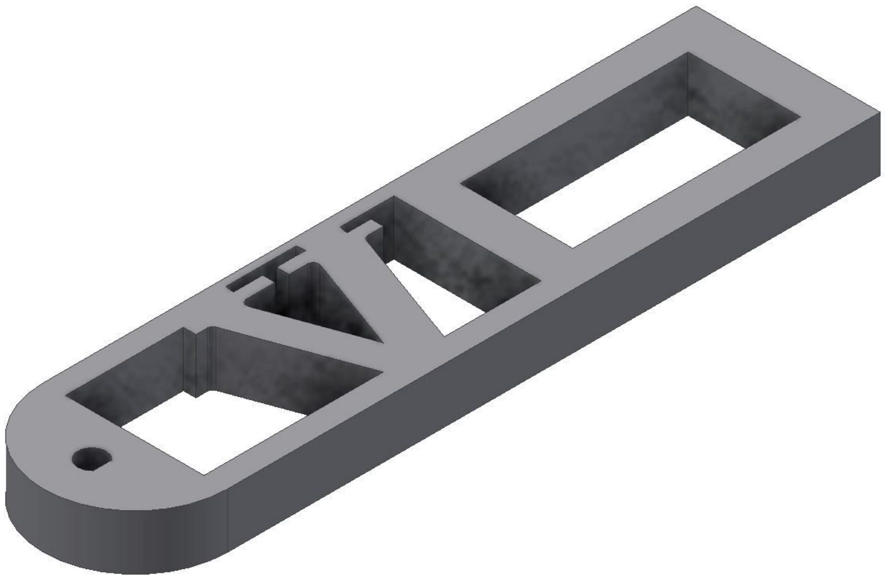
```
#include <Wire.h>
#include <Stepper.h>

//SERVO MOTOR Setup
int servoPin1 = 2; //accel up/down
int servoPin2 = 3; //tip rotator
int servoPin3 = 4; //cz claw

int pw1 = 1500;
int pw2 = 1500;
int pw3 = 1500;

int pwmax = 2100;
int pwmin = 900;

int period = 20;


//STEPPER MOTOR 1 Setup
//rotation
const int stepsPerRevolution1 = 100; // steps per revolution
// initialize the stepper library on pins 10-13:
Stepper myStepper1(stepsPerRevolution1, 6, 7, 8, 9);


//STEPPER MOTOR 2 Setup
//up/down

const int stepsPerRevolution2 = 100; // steps per revolution
// initialize the stepper library on pins 10-13:
Stepper myStepper2(stepsPerRevolution2, 10, 11, 12, 13);


//NUNCHUCK Setup
static uint8_t nunchuck_buf[6];   // array to store nunchuck data,

//////////////////////////////////////////////////////////////////
```

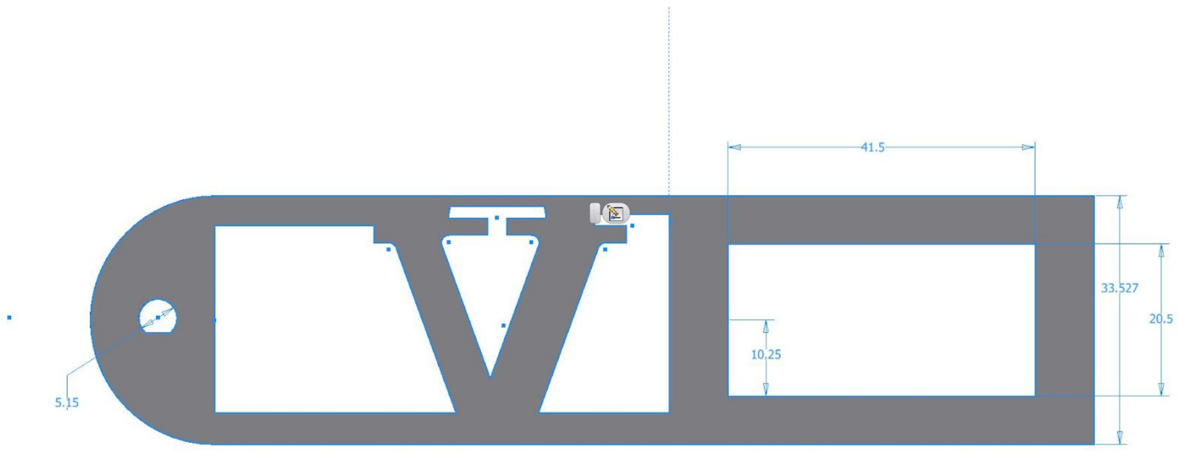
```
 void setup()
 {
  Serial.begin(19200);

  //SERVO MOTOR SETUP:
  pinMode(servoPin1, OUTPUT);
  pinMode(servoPin2, OUTPUT);
  pinMode(servoPin3, OUTPUT);
  nunchuck_setpowerpins(); // use analog pins 2&3 as fake gnd & pwr
  nunchuck_init(); // send the initilization

  //STEPPER MOTOR SETUP
  myStepper1.setSpeed(30); // set speed to 30 rpm
  myStepper2.setSpeed(30);
 }

///////////////////////////////////////////////////////////////////////

 void loop()
 {

  nunchuck_get_data();

  // map nunchuk data to a servo data point
  //int x_axis = map(nunchuck_buf[0], 23, 222, 180, 0);
  //int y_axis = map(nunchuck_buf[1], 32, 231, 0, 180);
  int x_axis = nunchuck_buf[0];
  int y_axis = nunchuck_buf[1];

  nunchuck_print_data();  //print data to serial monitor

  static int i=0;
  int joy_x_axis = nunchuck_buf[0];
  int joy_y_axis = nunchuck_buf[1];

  int accel_x_axis = nunchuck_buf[2];
  int accel_y_axis = nunchuck_buf[3];
  int accel_z_axis = nunchuck_buf[4];


  int z_button = 0;
  int c_button = 0;


///////////////////////////////////////////////////////////////////////

 // C AND Z BUTTON DATA
 if ((nunchuck_buf[5] >> 0) & 1)
  z_button = 1;
 if ((nunchuck_buf[5] >> 1) & 1)
  c_button = 1;


 // ACCELEROMETER DATA
 if ((nunchuck_buf[5] >> 2) & 1)
  accel_x_axis += 2;
 if ((nunchuck_buf[5] >> 3) & 1)
```

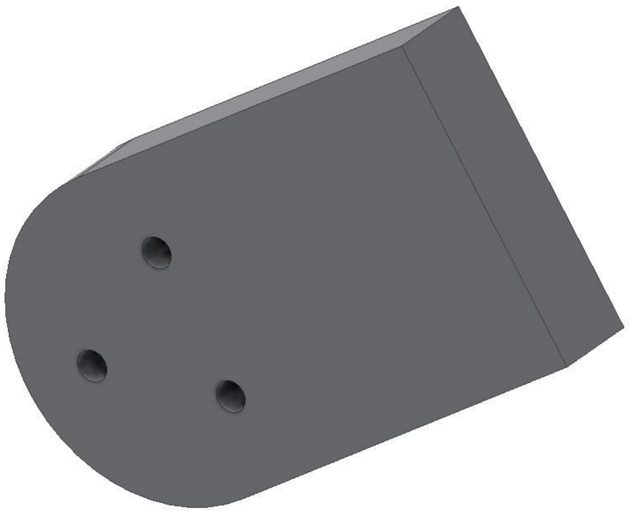
```
    accel_x_axis += 1;

  if ((nunchuck_buf[5] >> 4) & 1)
    accel_y_axis += 2;
  if ((nunchuck_buf[5] >> 5) & 1)
    accel_y_axis += 1;

  if ((nunchuck_buf[5] >> 6) & 1)
    accel_z_axis += 2;
  if ((nunchuck_buf[5] >> 7) & 1)
    accel_z_axis += 1;



////////////////////////////////////////////////////////////////////////


    //ARM ROTATOR STEPPER (joystick x-axis)

        //CW

        if (joy_x_axis < 110 && joy_x_axis > 30) {
           myStepper1.step(stepsPerRevolution1);
        }

        //CCW
        // step one revolution in the other direction:

        if (joy_x_axis < 230 && joy_x_axis > 140) {
          myStepper1.step(-stepsPerRevolution1);
        }

////////////////////////////////////////////////////////////////////////

    //ARM 1 STEPPER (joystick y-axis) UP/DOWN

         //CW

        if (joy_y_axis < 110 && joy_y_axis > 30) {
           myStepper2.step(stepsPerRevolution2);
        }

        //CCW

        if (joy_y_axis < 230 && joy_y_axis > 140) {
          myStepper2.step(-stepsPerRevolution2);
        }

////////////////////////////////////////////////////////////////////////

    //ARM 2 SERVO (accelerometer y-axis) UP/DOWN
        int pw1 = map(accel_y_axis, 70, 200, pwmin, pwmax);

        if( (accel_y_axis >70 && accel_y_axis<110) || (accel_y_axis>140 && accel_y_axis<200)) {
          digitalWrite(servoPin1, HIGH);
          delayMicroseconds(pw1);
```

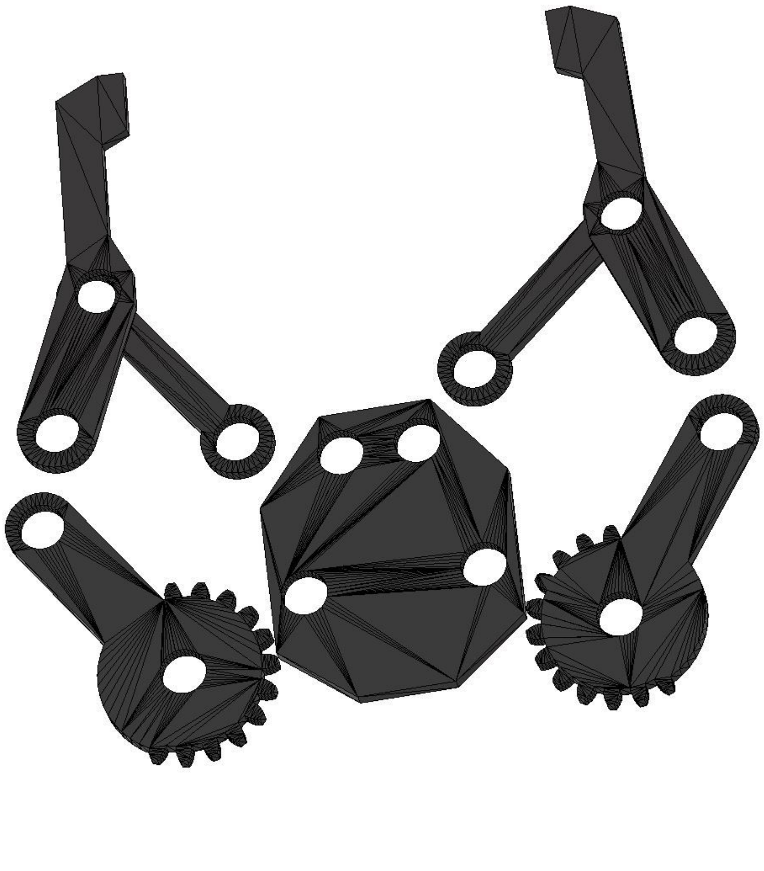
```
      digitalWrite(servoPin1, LOW);
      delay(period - pw1/ 1000);
    }

////////////////////////////////////////////////////////////////////////

  //END TIP ROTATOR (accelerometer x-axis)
    int pw2 = map(accel_x_axis, 70, 186, pwmin, pwmax);

    if( (accel_x_axis >70 && accel_y_axis<100) || (accel_y_axis>130 && accel_y_axis<186)) {
      digitalWrite(servoPin2, HIGH);
      delayMicroseconds(pw2);

      digitalWrite(servoPin2, LOW);
      delay(period - pw2/ 1000);
    }

////////////////////////////////////////////////////////////////////////

  // CLAW OPEN/CLOSE (c/z button)
    int gainVal = 150;
    if (pw3 <pwmin){
      pw3= pwmin;}

    if (pw3 >pwmax){
      pw3=pwmax;
    }

    if (pw3 <= pwmax && pw3 >=pwmin){
      if (c_button ==1 && z_button ==0) {
        pw3 = gainVal+pw3; }

      else if (z_button == 1 && c_button ==0) {
        pw3 = pw3- gainVal;}
    }

    digitalWrite(servoPin3, HIGH);
    delayMicroseconds(pw3);

    digitalWrite(servoPin3, LOW);
    delay(period - pw3 / 1000);
  }

////////////////////////////////////////////////////////////////////////


// NUNCHUCK FUNCTIONS

// Uses port C (analog in) pins as power & ground for Nunchuck
static void nunchuck_setpowerpins()
 {
 #define pwrpin PORTC3
 #define gndpin PORTC2
   DDRC |= _BV(pwrpin) | _BV(gndpin);
   PORTC &=~ _BV(gndpin);
   PORTC |=  _BV(pwrpin);
   delay(100);  // wait for things to stabilize
 }

// initialize the I2C system, join the I2C bus,
// and tell the nunchuck we're talking to it
void nunchuck_init()
 {
   Wire.begin();                  // join i2c bus as master
   Wire.beginTransmission(0x52);  // transmit to device 0x52
   Wire.write(0x40);              // sends memory address
   Wire.write(0x00);              // sends sent a zero.
   Wire.endTransmission();        // stop transmitting
 }

// Send a request for data to the nunchuck
// was "send_zero()"
void nunchuck_send_request()
 {
   Wire.beginTransmission(0x52);  // transmit to device 0x52
   Wire.write(0x00);              // sends one byte
   Wire.endTransmission();        // stop transmitting
 }

// Receive data back from the nunchuck,
int nunchuck_get_data()
{
   int cnt=0;
   Wire.requestFrom (0x52, 6);     // request data from nunchuck
   while (Wire.available ()) {
     // receive byte as an integer
     nunchuck_buf[cnt] = nunchuk_decode_byte(Wire.read());
     cnt++;
   }
   nunchuck_send_request();  // send request for next data payload
   // If we recieved the 6 bytes, then go print them
   if (cnt >= 5) {
    return 1; // success
   }
   return 0; //failure
}

// Print the input data we have received
// accel data is 10 bits long
// so we read 8 bits, then we have to add
// on the last 2 bits.  That is why I
// multiply them by 2 * 2

void nunchuck_print_data()
 {
   static int i=0;
   int joy_x_axis = nunchuck_buf[0];
   int joy_y_axis = nunchuck_buf[1];

   int accel_x_axis = nunchuck_buf[2]; // * 2 * 2;
   int accel_y_axis = nunchuck_buf[3]; // * 2 * 2;
   int accel_z_axis = nunchuck_buf[4]; // * 2 * 2;

   int z_button = 0;
   int c_button = 0;

 // byte nunchuck_buf[5] contains bits for z and c buttons
 // it also contains the least significant bits for the accelerometer data
 // check each bit of byte outbuf[5]

 // C AND Z BUTTON DATA
 if ((nunchuck_buf[5] >> 0) & 1)
   z_button = 1;
 if ((nunchuck_buf[5] >> 1) & 1)
   c_button = 1;

 // ACCELEROMETER DATA
 if ((nunchuck_buf[5] >> 2) & 1)
   accel_x_axis += 2;
 if ((nunchuck_buf[5] >> 3) & 1)
   accel_x_axis += 1;

 if ((nunchuck_buf[5] >> 4) & 1)
   accel_y_axis += 2;
 if ((nunchuck_buf[5] >> 5) & 1)
   accel_y_axis += 1;

 if ((nunchuck_buf[5] >> 6) & 1)
   accel_z_axis += 2;
 if ((nunchuck_buf[5] >> 7) & 1)
   accel_z_axis += 1;


// PRINT VALUES

   Serial.print(i,DEC);
   Serial.print("\t");

   Serial.print("joy x:");
   Serial.print(joy_x_axis,DEC);
   Serial.print(", y= ");
   Serial.print(joy_y_axis, DEC);
   Serial.print("  \t");

   Serial.print("acc x:");
   Serial.print(accel_x_axis, DEC);
   Serial.print(", y: ");
   Serial.print(accel_y_axis, DEC);
   Serial.print(", z: ");
   Serial.print(accel_z_axis, DEC);
   Serial.print("\t");

   Serial.print("z:");
   Serial.print(z_button, DEC);
   Serial.print(", c: ");
   Serial.print(c_button, DEC);

   Serial.print("\r\n");  // newline
   i++;
}
```

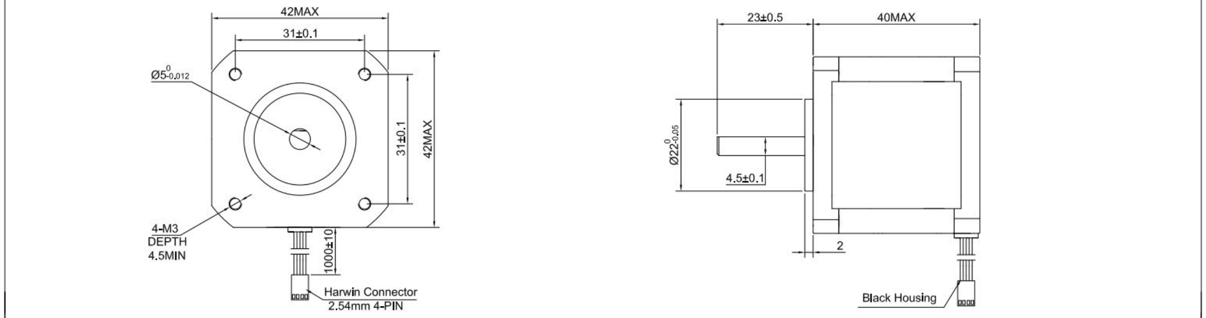
VI. *Appendix C: Datasheets and Drawings*











SPECIFICATION \ CONNECTION	BIPOLAR
VOTAGE(VOC)	2.20
AMPS/PHASE	2.00
RESISTANCE/PHASE(Ohms)@25°C	1.10±10%
INDUCTANCE/PHASE(mH)@1KHz	2.60±20%
HOLDING TORQUE(Nm)[lb-in]	0.45[3.98]
STEP ANGLE(°)	1.80
STEP ACCURACY(NON-ACCUM)	±5.00%
ROTOR INERTIA(g-cm²)	54.00
WEIGHT(Kg)[lb]	0.30[0.66]

TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）

AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]

INSULATION RESISTANCE 100 Mohm（UNDER NORMAL TEMPERATURE AND HUMIDITY）

INSULATION CLASS B 130°C[266°F]

DIELECTRIC STRENGTH 500VAC FOR 1MIN.(BETWEEN THE MOTOR COILS AND THE MOTOR CASE)

AMBIENT HUMIDITY MAX.85%(NO CONDENSATION)

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A —	BLK	A
2	A\ —	GRN	A\
3	B —	RED	B
4	B\ —	BLU	B\

FULL STEP 2 PHASE-Ex. ,
WHEN FACING MOUNTING END (X)

STEP	A	B	A\	B\		CCW
1	+	+	-	-	↓	↑
2	-	+	+	-		
3	-	-	+	+		
4	+	-	-	+	CW	

STEPPERONLINE
Motors&Electronics

	APVD		STEPPER MOTOR
	CHKD		
1:1	DRN		17HS16-2004S
SCALE	SIGNATURE	DATE	

FeeTech FS5103B - Standard Servo

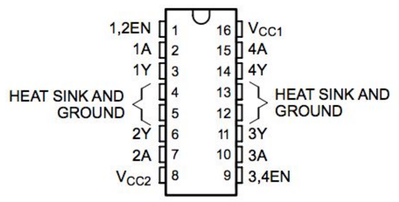
Specifications

Modulation:	Analog
Torque:	**4.8V:** 41.70 oz-in (3.00 kg-cm) **6.0V:** 44.50 oz-in (3.20 kg-cm)
Speed:	**4.8V:** 0.18 sec/60° **6.0V:** 0.16 sec/60°
Weight:	1.27 oz (36.0 g)
Dimensions:	**Length:** 1.61 in (40.8 mm) **Width:** 0.79 in (20.1 mm) **Height:** 1.50 in (38.0 mm)
Motor Type:	Brushed
Gear Type:	Plastic
Rotation/Support:	Dual Bearings
Rotational Range:	120°
Pulse Cycle:	(add)
Pulse Width:	900-2100 µs

Brand:	FEETECH
Product Number:	(add)
Typical Price:	(add)
Compare:	add+

5 Pin Configuration and Functions

NE Package
16-Pin PDIP
Top View



Pin Functions

PIN		TYPE	DESCRIPTION
NAME	**NO.**		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V_{CC1}	16	—	5-V supply for internal logic translation
V_{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

6 Specifications

6.1 Absolute Maximum Ratings

over operating free-air temperature range (unless otherwise noted)(1)

	MIN	MAX	UNIT
Supply voltage, V_{CC1} (2)		36	V
Output supply voltage, V_{CC2}		36	V
Input voltage, V_I		7	V
Output voltage, V_O	–3	V_{CC2} + 3	V
Peak output current, I_O (nonrepetitive, t ≤ 5 ms): L293	–2	2	A
Peak output current, I_O (nonrepetitive, t ≤ 100 µs): L293D	–1.2	1.2	A
Continuous output current, I_O: L293	–1	1	A
Continuous output current, I_O: L293D	–600	600	mA
Maximum junction temperature, T_J		150	°C
Storage temperature, T_{stg}	–65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) All voltage values are with respect to the network ground terminal.

6.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human-body model (HBM), per ANSI/ESDA/JEDEC JS-001(1)	±2000	V
		Charged-device model (CDM), per JEDEC specification JESD22-C101(2)	±1000	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

			MIN	NOM	MAX	UNIT
	Supply voltage	V_{CC1}	4.5		7	V
		V_{CC2}	V_{CC1}		36	
V_{IH}	High-level input voltage	V_{CC1} ≤ 7 V	2.3		V_{CC1}	V
		V_{CC1} ≥ 7 V	2.3		7	V
V_{IL}	Low-level output voltage		–0.3(1)		1.5	V
T_A	Operating free-air temperature		0		70	°C

(1) The algebraic convention, in which the least positive (most negative) designated minimum, is used in this data sheet for logic voltage levels.