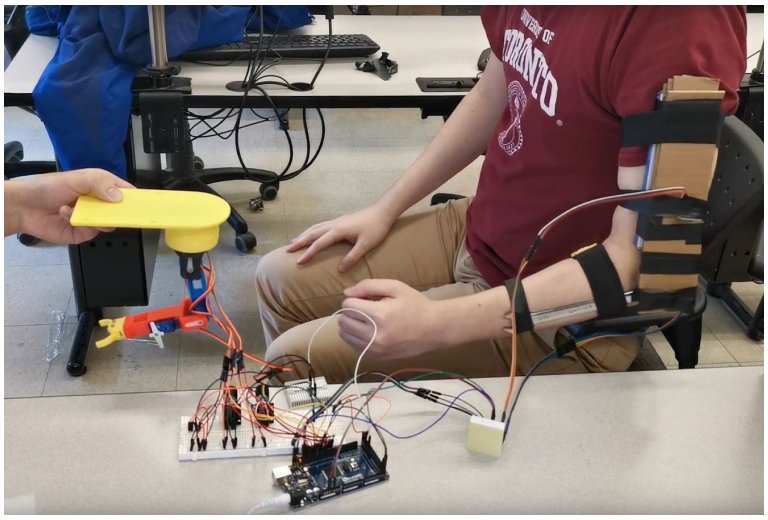
UNIVERSITY OF
TORONTO

University of Toronto

Faculty of Applied Science and Engineering

MIE 438 Microprocessors and Embedded Microcontrollers

Project Report



2019/04/11

Prepared by:

Hongzheng Xu	1001772904
Yixiao Hong	1001311145
Stephen Huang	1001165099
Ziqi Zhang	1001374684

1.0 Project Requirement

As the complexity and precision of robots evolve over the years, more and more industrial robots are employed to perform certain tasks. The objective of our project is to construct a 4 DoF robotic arm follower that can be easily controlled by a wearable device that attaches to a person's arm.

Originally, we chose the Arduino Uno with ATmega328P as our microcontroller due to its accessibility and flexibility. Arduino is an open-source electronics platform with relative easy programming and a large variety of accessories and libraries. These standard library sets and built-in functions are a major draw of the Arduino as it significantly simplifies the programming process. Another draw of the Arduino is that the team is quite familiar with the Arduino IDE and had many experiences programming on it. During the implementation, the Arduino Uno was replaced with Arduino Mega with the ATmega2560 microprocessor. One of the main reasons for the change is that the Mega has more interrupt and I/O pins compared to the Uno. More details are in section 4.0.

Despite the accessibility and flexibility, the Arduino Mega has its drawbacks on efficiency. The reduced complexity in programming is due to the excessive resource utilized and it would redundant in many cases. Also, the programming is c++ based which is considered high level, and it would not be as efficient as low-level machine language. In short, the Arduino Mega is already an embedded system rather than a single microprocessor. If we were to start the project over again, we would choose to program directly on a microprocessor such as one of the Atmel processors. It would better allow us to focusing on the basic level and optimize the overall design.

2.0 Project Design and Implementation

2.1 Overall Design Specification

The design goal is to develop a wearable device that can be attached to a person’s arm to capture and mimic the motion from the person’s arm. Human arm has 6 degrees of freedom as shown in FIgure 1, however, the team decided to simulate human arm with 4 degrees of freedom (shoulder pinch/ shoulder yaw and elbow pitch) plus a gripper to simulate the hand gripping action as these are the key representative motions of a human’s arm.

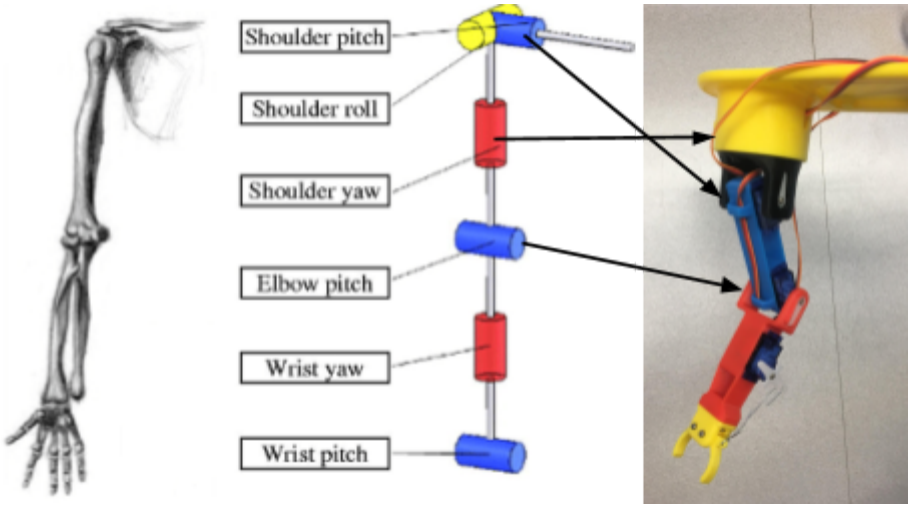


Figure 1: Comparison between Human arm and Our Robotic Arm

To achieve this, a gyroscope is attached to the person's arm, which senses the shoulder pitch and shoulder roll motion. An encoder is placed at the joint between arm and wrist to sense the elbow pitch motion. The sensor on the wearable device can capture the person's arm motion and convert it into the motor joints' positioning signals, which allows the robotic arm to present the same configuration as the person's arm. The robotic arm follower should also be able to perform simple pick and place operation under the control of a person.

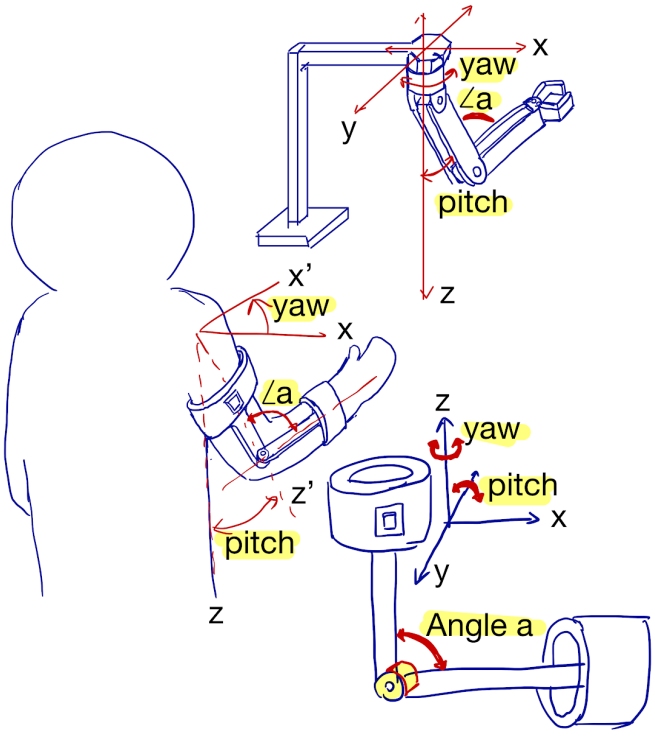


Figure 2: Conceptual Design

3.0 Component Selection

3.1 Frame

The robot frame is chose to be made by 3D printing with PLA material. The reason is that 3D printed parts provide a fast prototyping solution which reduce the testing and modification cycling time. In addition, PLA is a lightweight material that can easily be driven by the servo motor without needing much torque.

3.2 Actuators (Servo Motors)

The servo motor is chosen to be SG90 Micro Servo which provides an appropriate amount of torque (2.5kg-cm MAX) to drive the linkages [1]. It's working voltage is 5V which is a common supply voltage for most of embedded microcontroller boards.

3.3 Sensors:

The gyroscope is chosen to be MPU6050, which is the most popular portable gyroscope in the market. It is low cost and consume low power (3.3V). It can measure three axial rotary motion and with the built in A/D converter, it can generate digital signal output, which makes the data handling easy. The MPU6050 uses I2C-bus as the serial protocol, which is a very common serial protocol which only requires two digital inputs for the microcontroller board.

The rotary encoder used in this project is KY-040, which is a common portable encoder in the market. It requires a power supply of 5V DC and two digital inputs to sense the clockwise/counterclockwise rotary signals. It has a relatively low resolution of only 40 pusles/ rev, which corresponds to 8 degrees per pulse. However, its resolution is acceptable for our project as a demo and can be further improved by changing into a higher resolution encoder.

The following table shows a summary of the components used in this project (excluding the microcontroller)

Table 1. System Components

Item	Item Number	Required I/O Pin(s)	Quantity
Gyroscope	MPU6050	3.3V, GND, SCL, SDA, Interrupt Pin	1
Encoder	KY-040	5V, GND, CLK (Digital in, with interrupt), DT (Digital in, with interrupt)	1
Servo Motor	SG90	5V, GND, Digital Out (PWM)	4
Button	N/A	5V, GND, Digital In (with interrupt)	1

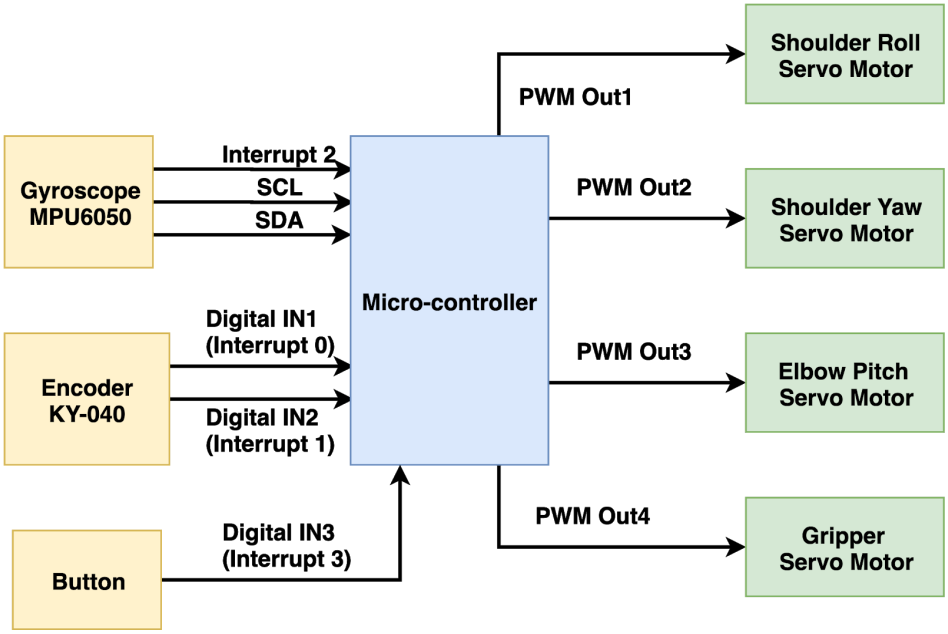


Figure 3: System I/O Configuration

4.0 Microcontroller Selection

Based on the functionality of the design, we determined that Arduino Mega with ATmega2560 would be suitable for the project. There are some obvious advantages as following:

- Simplicity: The Arduino Mega uses C++ based programming language, which reduces the complexity significantly for a project of our size when compared to low level machine language. Also, the Arduino has a user friendly IDE and many pre-build functions that are ready to use, which greatly increases the programming efficiency. Further, the hardware component setup and control using Arduino is simple and straightforward. Moreover, the Arduino is an open-source platform and therefore there are various of library ready online to use along with examples and demonstrations.

- Peripheral Compatibility: Both I2C and SPI protocols are supported. Communication lines are pre-defined for easy connection.

- I/O: Arduino Mega has 54 GPIO and 15 PWM output. Sufficient amount of GPIO and PWM can ease the testing and debugging for the project since it allows the team to not have to frequently change the connection when switching between different configurations. For the purpose of accuracy, we need interrupt on the gyroscope and encoder. Arduino Mega has 8 interrupt pins, which is desirable for our project since we need 1 for the gyroscope and 2 for the encoder and 1 for the button.

- Clock Speed: The most demanding component in terms of clocking speed is the MPU6050 gyroscope. It requires a minimum sampling frequency of 8kHz and the I2C bus requires a frequency of 500 kHz. The Arduino Mega has a 16MHz clock rate, which allows a decent response time, this is especially important considering the functionality (arm follower) and when we have interpret applied.
- Power: Arduino Mega can output 3.3V or 5V which is compatible with the peripherals so there is no extra external power supply or DC-DC converter required. It also has built-in power regulator which makes it more reliable.

4.1 Programming Language and Code Optimization

Due to the complexity of the overall logic, the program is written in high-level language (c++) instead of assembly language. The overall program flow can be seen in the figure below.

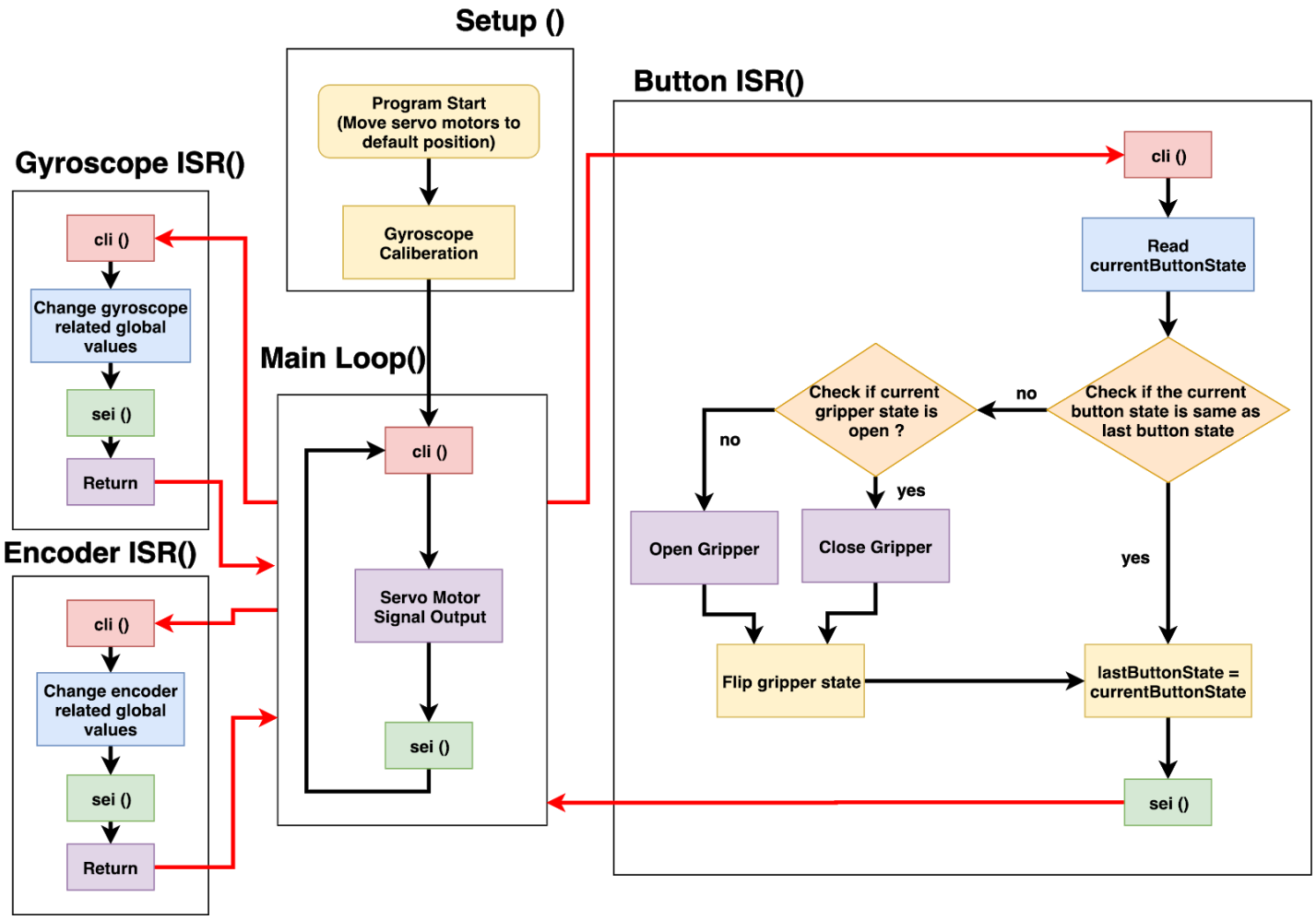


Figure 4: Program Flow Diagram

With the flow of the memory at the assembly level in mind, the high-level code is optimized using several techniques mentioned in class. The following code is an example of sub-expression elimination.

On the right side is the encoder debouncing module, which is inside the ISR that update the encoder angle. Variables that are used multiple times in line 9 to line 36 are pre-calculated in line 3 to line 7. By sub-expression elimination, the execution speed of the ISR is improved. Consequently, the maximum rotational speed allowance of the encoder is increased by 50%.

```
1  void mid_encoder_read() {
2  cli();
3    DT=digitalRead(PinDT);
4    CLK=digitalRead(PinCLK);
5    allDTp=allDT[count+1];
6    allCLKp=allCLK[3-count];
7    allDTm=allDT[3-count];
8
9    if (count==0){
10     if(DT==allDTp&&CLK==allCLKp){
11       direc = 1;
12       count++;
13     }
14     else if(DT==allDTm&&CLK==allCLKm){
15       direc = 2;
16       count++;
17     }
18   }
19   if(DT==allDTp&&CLK==allCLKp&&direc==1){   //c
20     if(count==3) {
21         count=0;
22         direc=0;
23         virtualPosition++;
24         Serial.println(virtualPosition);
25     }
26     else if(count<3) count++;
27   }
28   else if(DT==allDTm&&CLK==allCLKm&&direc==2){
29     if(count==3) {
30         count=0;
31         direc=0;
32         virtualPosition--;
33         Serial.println(virtualPosition);
34     }
35     else if(count<3) count++;
36   }
37   sei();
38 }
39
```

Figure 5. Sub-expression Elimination Example

4.2 Interrupt Service Routine

At the beginning of the project, all the sensor-reading functions are called in the main loop. Sensor values are checked during every single run of the main loop. However, it quickly comes to the team's attention that as more instructions are executed in the main loop, the looping time of the main loop increases dramatically. As the looping frequency of the main loop drops, there is an increased chance that sensor outputs are ignored by the program, which leads to inaccuracy. In order to solve this issue, interrupt service routine (ISR) is utilized. With the built-in interrupt pins (pin 2, 3, 18, 19, 20, 21) of the Arduino Mega, the sensor readings are able to be updated with minimal impact by the looping frequency of the main loop. Moreover, by utilizing ISR, a lot of memory resources are saved, while the load on the CPU is greatly reduced.

4.3 Encoder Debouncing

The encoder used in this project, KY-040, is a conductive encoder. The basic components of a conductive encoder includes two copper pins of different lengths and a disk that rotates about its central axis. A series of circumferential copper tracks are attached to the disk. When the disk rotates, the two pins that are in contact with the tracks create asynchronous electrical pulses. By identifying the pattern of the pulses, the direction and the speed of the encoder's rotation are determined.

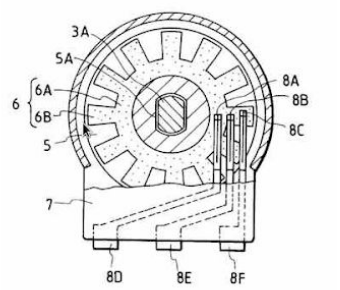


Figure 6. Rotary Encoder Assembly [2]

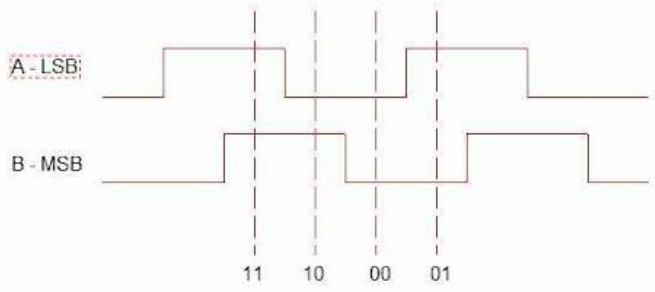


Figure 7. Ideal Rotary Encoder Pulse [2]

Due to the nature of the encoder's mechanical system, the contact pins tend to bounce, which results in noisy signals. To overcome the noise in order to obtain an accurate reading, a debouncing module that utilizes a finite state machine approach is developed.

As shown in figure 7, different combinations of A-LSB and B-MSB are categorized into 4 states, 11, 10, 00, 01. Each state is associated with transitions that check if the next reading belongs to either the state to the left or the state to the right. If the condition is met, the output of the transition will set the current state to the next state. Otherwise, the output of the transition will discard the invalid reading and remain the current state. The finite state machine approach can effectively filter out the switch bounce.

4.4 Button Debouncing

The button used in this project is a simple button that when pressed, sends a 5V signal to one of Mega's input pin. We created a simple boolean flag that determines the if the gripper is opened or closed. When the button is pressed down, the Mega will read a "high" signal. Originally, the plan was to change the flag's status whenever the Mega reads a "high" signal from the input pin, however, we soon discovered a issue which is button bouncing. Specifically, when the button is held in the pressed position, the Mega would constantly read a "high" signal. This causes the gripper to open and close continuously. We solved this issue by changing the program so that the flag's status gets changed when the Mega's input pin goes from "high" to "low".This new setup

means the gripper would either open or close only once every time the button is pressed down and released.

5.0 Project Result

At the end, the completed design works quite well as it can successfully follow a person's arm movements with decent speed and accuracy. Please refer to the showcase video file in order to see the robotic arm in action. After implementing all the previously stated optimizations, our robotic arm is capable of following joint movements up to the speed of 60 rpm.

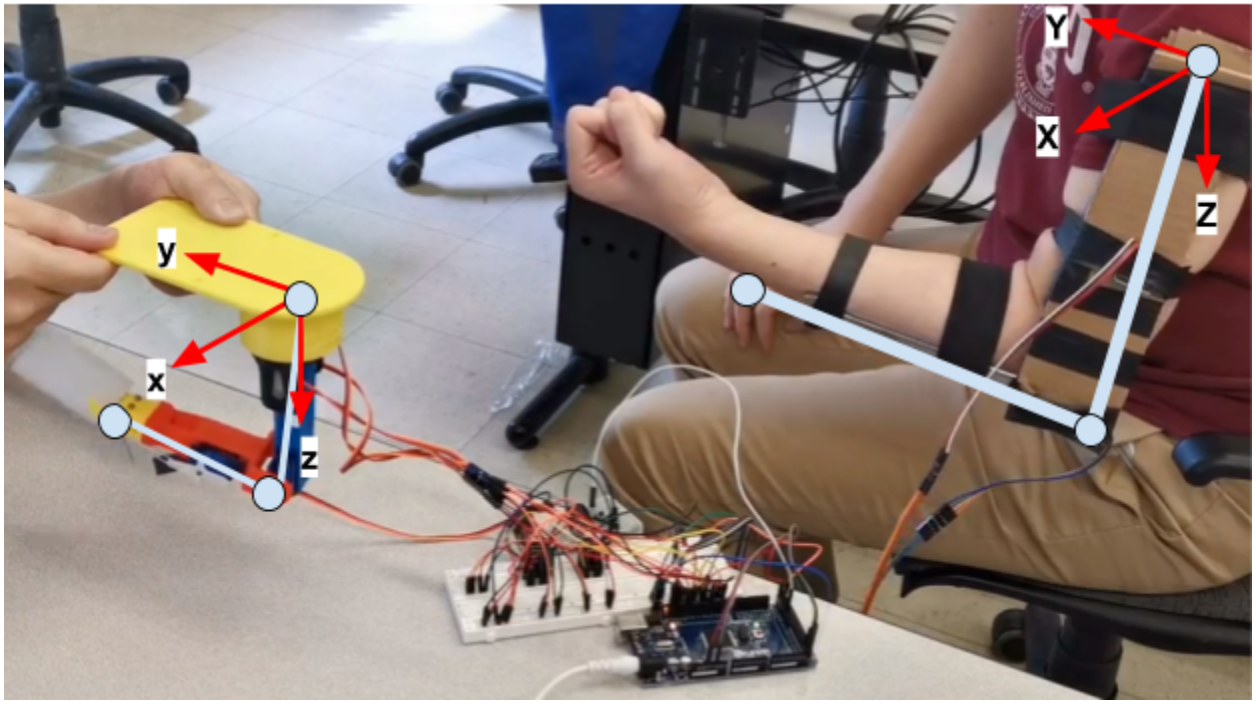


Figure 8: Project Final Result

6.0 Future improvements

1. As stated in section 4.0, ATmega2560 supports the direct plug-in of various types of peripherals. For the purpose of easy connection, a lot of features are embedded into the processor. Although it is good for general purpose, small scale projects, some features might not be useful for specific tasks. In our project, both the input and the output are digital. Therefore, the built in ADC and DAC converter are not utilized at all. The optimized solution for this project is to use a processor that does not natively support ADC, which may result in a cheaper price.

2. The ATmega2560 natively contains 6 interrupt pins, which is more than enough for this project. However, the code size that can be fit into a single ISR is greatly restricted by the clocking speed of the processor. In order to filter out the noise in the encoder reading, a relatively complex debouncing logic is required to be fit into the ISR. If the interrupt with large amount of code is called too frequently, some interrupts will be dismissed unintentionally. Thus, the maximum allowed rotational speed of the encoder is restricted. This issue can be mitigated by switching to a processor with higher clock speed. For example, Raspberry Pi 3 has the base clock speed of 1.2GHz [3]. It is much faster than the base clock speed for the ATmega2560, which is only 16MHz. However, it must be kept in mind that faster clocking speed always associates with higher cost.

3. Currently, there is not PID controller involved in the system. Adding a PID controller will improve the overall accuracy and responsiveness of the robotic arm. Note: A lot of masking is involved in during the calculation of the PID controller. The section to be masked must be carefully considered since excessive masking might cause precision lost on the sensors.

4. Arduino Atmel AVR Instruction Set is easy to access but gives less freedom for low-level modification.

5. Due to the complexity of the gyroscope calculation, the best solution will be to choose a microprocessor that can handle multi-thread processing. With sensors run on seperate threads, accuracy can be guaranteed since unintentionally masking of ISR is prevented. ATmega2560 does natively support multi-thread processing. A solution is to switch to another processor that support multi-thread processing. An example will be the Raspberry Pi 3. Another solution is to implement Protothreading, which is used on small-scale microprocessor . Protothreading simulates the workflow of a multi-thread processor. It allows simple codes to be run in parallel.

6. Use optical encoder, less debouncing needed, lighter code in ISR, improve encoder rotational speed allowance

7.0 Reference

[1] P.Y.K. Cheung, “SERVO MOTOR SG90”, *Imperial College London*, 2018. [Online].

Available:http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

[2] Best Microcontroller Projects, “Types of Rotary Encoder,”*best-microcontroller-projects.com*, 2018[Online].Available:https://www.best-microcontroller-projects.com/rotary-encoder.html#Taming_Noisy_Rotary_Encoders

[3] Magpi Magazine, “RASPBERRY PI 3: SPECS, BENCHMARKS & TESTING,”*www.raspberrypi.org*, 2016[Online].Available:https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

Appendix A

Arduino Code

```
//-----------------------------------------------//

//--------------------Library--------------------//

//-----------------------------------------------//

#include <Servo.h>

#include <Wire.h>

#include <SoftwareSerial.h>


#define gyroInterrupt 18;


//-----------------------------------------------//

//------------------Variables--------------------//

//-----------------------------------------------//


//--------------------Gyros--------------------//
```

```
long accelX, accelY, accelZ;
float gForceX, gForceY, gForceZ;

long gyroXCalli = 0, gyroYCalli = 0, gyroZCalli = 0;
long gyroXPresent = 0, gyroYPresent = 0, gyroZPresent = 0;
long gyroXPast = 0, gyroYPast = 0, gyroZPast = 0;
float rotX, rotY, rotZ;

float angelX = 0, angelY = 0, angelZ = 0;

int angleZ = 0;
int angleX = 0;
int angleY = 0;

long timePast = 0;
long timePresent = 0;

int gyroServoBase = 0;
int gyroServoRoot = 0;

//----------------------Servos----------------------//
Servo baseServo;  // create servo object to control a servo
int baseServoSingal = 0;
```

```
Servo rootServo;
int rootServoSingal = 0;

Servo midServo;
int midServoSingal = 0;

Servo midServo_gyro;
int midServoSingal_gyro = 0;

Servo gripperServo;
int gripperMoterSingnal = 90;

int servoOut = 0;
//----------------------Servo Variables----------------------//
int baseServoOut;
int rootServoOut;
int midServoOut;
int gripperServoOut;

//Gyro_Mid
int encoderPosCount_mid_gyro = 0;
int correctEncoderPosCount_mid_gyro = 0;
int aVal_mid_gyro;
boolean bCW_mid_gyro;
```

```
//----------------------Encoder Variables----------------------//

#define PinCLK 2

#define PinDT 3


static long virtualPosition=0;    // without STATIC it does not count correctly!!!


bool CLK = false;

bool DT = false;

bool allCLK[5] = {1, 0, 0, 1, 1};

bool allDT[5] = {1, 1, 0, 0, 1};

int count = 0;

int direc = 0;  //0 not known, 1 cw, 2 ccw

int countPlus, minusCount;


//----------------------Button VARIABLES----------------------//

int buttonPin = 19;

int buttonState = 0; //open

int last_button = 0;

int flag = 0;  //open state



//------------------------------------------------------//

//----------------------Set Up-------------------------//
```

```
//--------------------------------------------------//

void setup() {
  //Servo
  baseServo.attach(11);  // attaches the servo on pin 11 to the servo object
  rootServo.attach(10);  // attaches the servo on pin 10 to the servo object
  midServo.attach(9);  // attaches the servo on pin 9 to the servo object
  gripperServo.attach(6);

  //button
  pinMode(buttonPin,INPUT);


  //Initial Servo Position
  baseServo.write(76);
  rootServo.write(54);
  midServo.write(18);
  gripperServo.write(map(10, 0, 20, 0, 180));

  //Serial
  Serial.begin (9600);

  //GyroScope Set Up
  Wire.begin();
```

```
  setUpMPU();
  Serial.println("Start Cali");
  callibrateGyroValues();

  timePresent = millis();
  Serial.println("Finish Cali");

  //Encoder Interrupt Setting
  pinMode(CLK,INPUT);
  pinMode(DT,INPUT);
   attachInterrupt (0,mid_encoder_read,CHANGE);    // interrupt 0 is always connected to pin 2 on Arduino UNO
  attachInterrupt (1,mid_encoder_read,CHANGE);

  //GyroScope Interrupt Setting
  attachInterrupt (2,gyro_ISR,CHANGE);  //interrupt 2 is pin18

  //Button Interrupt Setting
  attachInterrupt (3,button_ISR,CHANGE);

}

//------------------------------------------------//
//----------------------Main----------------------//
//------------------------------------------------//
```

```
void loop() {

  cli();

  gyro_servoSignalOut();

  sei();

  delay (100);
}

//-------------------------------------------------//
//-----------------ISR Functions-------------------//
//-------------------------------------------------//



void gyro_ISR (){

  cli();
  Read_Gyro();   //Read Gyroscope
  sei();
```

```
}

void encoder_ISR (){

  cli();
  mid_encoder_read();  //Read Encoder
  sei();

}

void button_ISR(){

  cli();

  buttonState = digitalRead(buttonPin);

  if (buttonState < last_button){
     flag = 1 - flag;
   }

  if (flag == 0 ){
   gripperOpen();
  }
```

```
  if (flag == 1){
    gripperClose();
  }

  last_button = buttonState;

  sei();

}

//------------------------------------------------//
//---------------------Functions---------------------//
//------------------------------------------------//


//---------------------Encoder Correction Function---------------------//
//Corrent final reading when the encoder reading exceed 0-360 degree
int correctEncoder(int input){
    int output = input % 40;
    if (output<0){
      output = output + 40;
    }
    return output;
 }
```

```
//----------------------Encoder Half Control Function----------------------//

//Make the encoder output mirror to the 0-180 reading since the servo motor can only rotate 0-180 degree

int gyro_mid_halfcorrection_encoder(int input, int error){

   input = input*9;

   int output;


   if (input>=0 && input <= 180 - error){

     output = input + error;

   }

   else if (360- error<=input && input<360){

     output = input - (360-error);

   }

   else {

     output = 360-error - input;

   }


   return output;


}



//----------------------Gyro Mid Encoder Read Function----------------------//
```

```
void mid_encoder_read() {
cli();
  DT=digitalRead(PinDT);
  CLK=digitalRead(PinCLK);
  countPlus=count+1;
  minusCount=3-count;

  if (count==0){
    if(DT==allDT[countPlus]&&CLK==allCLK[countPlus]){
      direc = 1;
      count++;
    }
    else if(DT==allDT[minusCount]&&CLK==allCLK[minusCount]){
      direc = 2;
      count++;
    }
  }
  if(DT==allDT[countPlus]&&CLK==allCLK[countPlus]&&direc==1){   //cw
    if(count==3) {
          count=0;
          direc=0;
          virtualPosition++;
          Serial.println(virtualPosition);
    }
```

```
    else if(count<3) count++;
  }
  else if(DT==allDT[minusCount]&&CLK==allCLK[minusCount]&&direc==2){    //ccw
   // Serial.print(allDT[minusCount]);
    if(count==3) {
          count=0;
          direc=0;
          virtualPosition--;
          Serial.println(virtualPosition);
    }
    else if(count<3) count++;
  }
  sei();
}


void gripperOpen(){
  //mid_halfcorrection_encoder(correctEncoderPosCount_mid);
  servoOut = map(10, 0, 20, 0, 180);     // scale it to u se it with the servo (value between 0 and 180)
  gripperServo.write(servoOut);                 // sets the servo position according to the scaled value

}

void gripperClose(){
```

```
  //mid_halfcorrection_encoder(correctEncoderPosCount_mid);
  servoOut = map(0, 0, 20, 0, 180);     // scale it to u se it with the servo (value between 0 and 180)
  gripperServo.write(servoOut);                 // sets the servo position according to the scaled value

}



//----------------------Servo Reading Correction Function----------------------//
int base_correction_gyro(int input, int error){
    int output;

    if (input>=0 && input <=error){
      output = -input + error;
    }
    else if (error+180<=input && input<360){
      output = 360 + error - input;
    }
    else {
      output = input - error;
    }

    return output;

}
```

```
int root_correction_gyro(int input, int error){
   int output;

   if (input>=0 && input <=180-error){
     output = input + error;
   }
   else if (360-error<=input && input<360){
     output = error + input -360 ;
   }
   else {
     output = 360-error-input;
   }

   return output;

}
```

```
//----------------------Gyro Servo Siugnal Out----------------------//
void gyro_servoSignalOut_mid(){
                                                    midServoSingal_gyro                    =
gyro_mid_halfcorrection_encoder(correctEncoderPosCount_mid_gyro,27);
```

```
  servoOut = map(midServoSingal_gyro, 0, 180, 0, 180);     // scale it to u se it with the servo (value between 0 and 180)
  midServo.write(servoOut);                 // sets the servo position according to the scaled value
}

void gyro_servoSignalOut(){
  int gyroBaseError = 76;
  gyroServoBase = base_correction_gyro(angleZ,gyroBaseError);
  baseServoOut = map(gyroServoBase, 0, 180, 0, 180);      // scale it to u se it with the servo (value between 0 and 180)
  baseServo.write(baseServoOut);                // sets the servo position according to the scaled value

  int gyroRootError = 70;
  gyroServoRoot = root_correction_gyro(angleX,gyroRootError);
  rootServoOut = map(gyroServoRoot, 0, 180, 0, 180);      // scale it to u se it with the servo (value between 0 and 180)
  rootServo.write(rootServoOut);                // sets the servo position according to the scaled value

  gyro_servoSignalOut_mid();
}

//----------------------Gyro Functions----------------------//
void setUpMPU() {
```

```
  // power management
  Wire.beginTransmission(0b1101000);        // Start the communication by using address of MPU
  Wire.write(0x6B);                         // Access the power management register
  Wire.write(0b00000000);                   // Set sleep = 0
  Wire.endTransmission();                   // End the communication

  // configure gyro
  Wire.beginTransmission(0b1101000);
  Wire.write(0x1B);                         // Access the gyro configuration register
  Wire.write(0b00000000);
  Wire.endTransmission();

  // configure accelerometer
  Wire.beginTransmission(0b1101000);
  Wire.write(0x1C);                         // Access the accelerometer configuration register
  Wire.write(0b00000000);
  Wire.endTransmission();
}

void callibrateGyroValues() {
   for (int i=0; i<5000; i++) {
     getGyroValues();
     gyroXCalli = gyroXCalli + gyroXPresent;
     gyroYCalli = gyroYCalli + gyroYPresent;
```

```
      gyroZCalli = gyroZCalli + gyroZPresent;
    }
    gyroXCalli = gyroXCalli/5000;
    gyroYCalli = gyroYCalli/5000;
    gyroZCalli = gyroZCalli/5000;
}

void readAndProcessAccelData() {
  Wire.beginTransmission(0b1101000);
  Wire.write(0x3B);
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6);
  while(Wire.available() < 6);
  accelX = Wire.read()<<8|Wire.read();
  accelY = Wire.read()<<8|Wire.read();
  accelZ = Wire.read()<<8|Wire.read();
  processAccelData();
}

void processAccelData() {
  gForceX = accelX/16384.0;
  gForceY = accelY/16384.0;
  gForceZ = accelZ/16384.0;
}
```

```
void readAndProcessGyroData() {
  gyroXPast = gyroXPresent;                    // Assign Present gyro reaging to past gyro reading
  gyroYPast = gyroYPresent;                    // Assign Present gyro reaging to past gyro reading
  gyroZPast = gyroZPresent;                    // Assign Present gyro reaging to past gyro reading
  timePast = timePresent;                     // Assign Present time to past time
   timePresent = millis();                          // get the current time in milli seconds, it is the present time

  getGyroValues();                          // get gyro readings
  getAngularVelocity();                       // get angular velocity
  calculateAngle();                         // calculate the angle
}

void getGyroValues() {
   Wire.beginTransmission(0b1101000);                  // Start the communication by using address of MPU
  Wire.write(0x43);                          // Access the starting register of gyro readings
  Wire.endTransmission();
   Wire.requestFrom(0b1101000,6);                  // Request for 6 bytes from gyro registers (43 - 48)
  while(Wire.available() < 6);                  // Wait untill all 6 bytes are available
  gyroXPresent = Wire.read()<<8|Wire.read();          // Store first two bytes into gyroXPresent
  gyroYPresent = Wire.read()<<8|Wire.read();          // Store next two bytes into gyroYPresent
  gyroZPresent = Wire.read()<<8|Wire.read();          //Store last two bytes into gyroZPresent
```

```
}

void getAngularVelocity() {
  rotX = gyroXPresent / 131.0;
  rotY = gyroYPresent / 131.0;
  rotZ = gyroZPresent / 131.0;
}

void calculateAngle() {
  // same equation can be written as
   // angelZ = angelZ + ((timePresentZ - timePastZ)*(gyroZPresent + gyroZPast - 2*gyroZCalli)) / (2*1000*131);
  // 1/(1000*2*131) = 0.00000382
  // 1000 --> convert milli seconds into seconds
  // 2 --> comes when calculation area of trapezium
  // substacted the callibated result two times because there are two gyro readings
   angelX = angelX + ((timePresent - timePast)*(gyroXPresent + gyroXPast - 2*gyroXCalli)) * 0.00000382;
   angelY = angelY + ((timePresent - timePast)*(gyroYPresent + gyroYPast - 2*gyroYCalli)) * 0.00000382;
   angelZ = angelZ + ((timePresent - timePast)*(gyroZPresent + gyroZPast - 2*gyroZCalli)) * 0.00000382;
}

void Read_Gyro(){
  readAndProcessAccelData();
```

```
readAndProcessGyroData();
angleZ = angelZ;
angleZ = angleZ % 360;
if (angleZ > 0 ){
  angleZ = 360 - angleZ;
}
else {
  angleZ = - angleZ;
}

angleX = angelX;
angleX = angleX % 360;
if (angleX > 0 ){
  angleX = 360 - angleX;
}
else {
  angleX = - angleX;
}

angleY = angelY;
angleY = angleY % 360;
if (angleY > 0 ){
  angleY = 360 - angleY;
}
```

```
else {
  angleY = - angleY;
}

//Serial.println(angleZ);
//Serial.println(angleX);
//Serial.println(angleY);

}
```