5-2024

Rock-Paper-Scissors Robot

Cora Lewis
Trinity University

Zachary Moyer
Trinity University

Aakriti Acharya
Trinity University

Reece Colson
Trinity University

Rock-Paper-Scissors Robot Final Report

ENGR-4367
Instructor: Dr. Kevin Nickles
Project Group 3: Cora Lewis, Zachary Moyer, Aakriti Acharya, and Reece Colson
Pledged
April, 30th 2024

Table of Contents

1 Design Summary

The design incorporates a sophisticated Rock Paper Scissors (RPS) playing interface, amalgamating robotics, audio control, gesture recognition, and display functionalities. At its core, the system employs a robotic hand mechanism featuring pulleys and fishing wire to simulate finger movements. This mechanism is orchestrated by servo motors through a PIC system, which receives commands from the Arduino. The Arduino serves as the central processor and begins a round when the player hits the reset button. It triggers an audio prompt at the onset of each round using an Icstation Recordable Sound Module, prompting the player to make their move. The Arduino then selects the RPS move based on the player's gesture captured via flex sensors integrated into their glove. These sensors, acting as variable resistors, detect finger movements and interface with the Arduino through a voltage divider circuit and ADC. The system operates in two modes: easy and hard. In easy mode, the move selection is random, while in hard mode, a rational agent determines the optimal move. Once both the player and the robotic hand have made their moves, the Arduino determines the winner and updates the scores on an LCD connected via I2C communication. This comprehensive integration of hardware components and intelligent algorithms offers an engaging and interactive RPS gaming experience, combining physical interaction with digital feedback seamlessly.

2 System Details

2.1 Robotic Hand

The robotic hand consists of a series of pulleys with two lines of fishing wire threaded through each finger. When the top line is pulled, the tension in the wire extends the finger, as shown in Figure # in the appendix. Similarly, when the inverse bottom line is pulled, the finger contracts due to the tension, as shown in Figure Y in the appendix. These two lines are attached to opposite ends of a servo mount, which allows a servo to open and close the fingers by spinning in different directions. The full-hand system is illustrated in Figure 1.

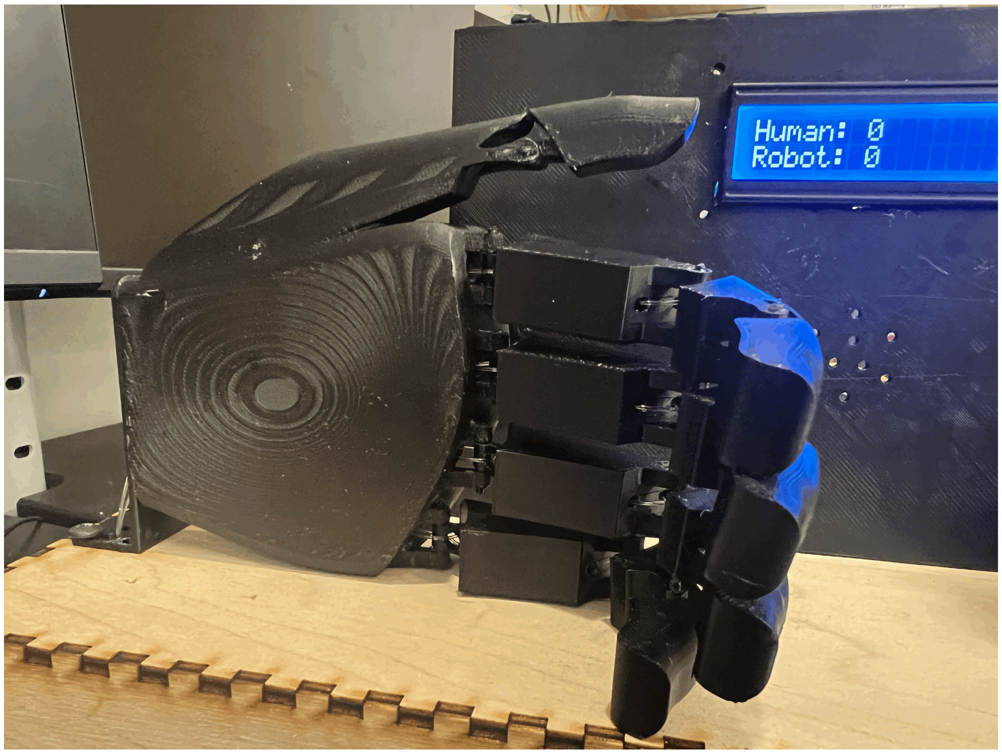

Figure 1: PLA moveable robotic hand

2.2 PIC System

The PIC system primarily serves to control two continuous rotation servo motors. These servo motors operate with an internal clock set at 50Hz. To generate the required pulse width modulation signal, we employed a 4 MHz internal oscillator, coupled with some testing that determined approximately 5% and 12% duty cycles as optimal for achieving the desired rotation speed and distance for the servos.

In essence, the PIC system activates upon receiving a start signal from the Arduino, followed by a command specifying the Rock Paper Scissors (RPS) move to be executed. Using this information, it directs the two servos to move in the appropriate directions. This process is outlined in the program flowchart shown in Figure 2. One servo governs the pointer and middle fingers, while the other manages the ring finger and pinky.

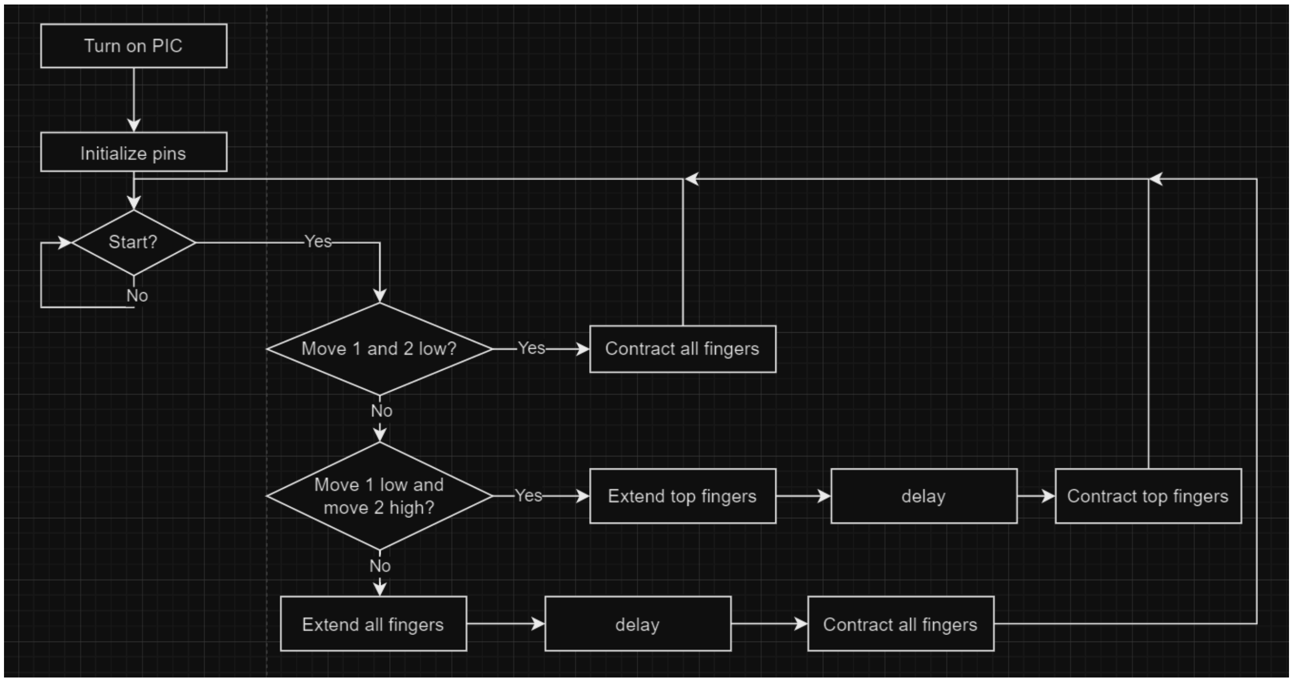

Figure 2: PIC Program Flowchart

2.3 Arduino System

2.3.1 Audio Control

The Arduino system acts as the main processor of the system. When the player initiates the round, the Arduino starts an audio file on an Icstation Recordable Sound Module which plays Dr. Nickels saying "Rock Paper Scissors" and signals to the player to toss whatever move they choose to toss once the audio ends. The module comes with three surface-mounted resistors which determine what inputs to the input pin will cause the audio to play [1]. To integrate this speaker into the circuit the op-amp buffer circuit illustrated in Figure 3 is utilized to sink current.

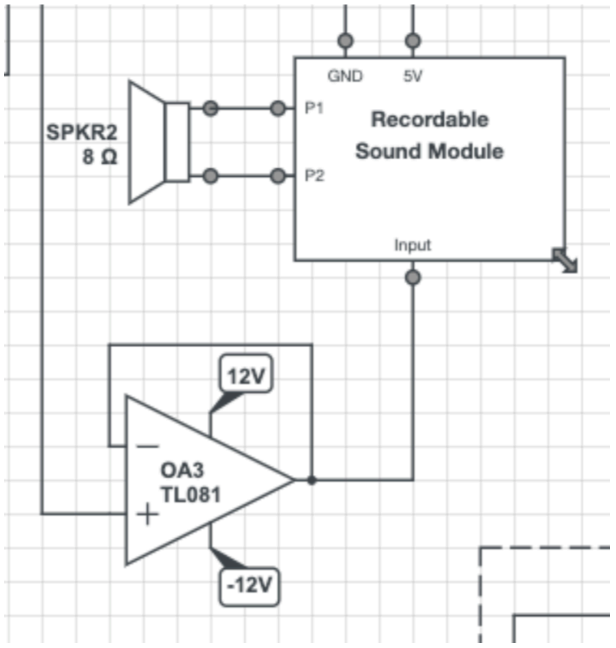

Figure 3: ICstation speaker with integrated MP3 file support and input buffer

2.3.2 Move Calculation

While the audio file is playing, the Arduino selects the rock-paper-scissors move to execute and then transmits it to the PIC with three output pins. The first two wires transmit the moves and the third triggers the PIC to play the move. An input switch, known as the hard mode switch, selects the algorithm for picking the next move. This switch toggles between easy and hard modes.

In easy mode, the random function is invoked on numbers ranging from 0 to 2, with each number corresponding to a particular RPS move. In hard mode, a rule-based rational agent picks the next move based on the previous moves thrown. If the robot won the previous move, then it selects the same move. If the robot lost, then it picks the move that would have won the previous round.

2.3.3 Flex Sensors

After the audio file concludes, the Arduino interprets the player's gestures by analyzing input from two flex sensors integrated into the player's glove, as illustrated in Figure 4. These flex sensors function as variable resistors, with their resistance fluctuating between 20kΩ and 50kΩ in response to bends and stretches. By incorporating them into a voltage divider circuit, a microcontroller can gauge the degree of finger bending using an analog-to-digital converter (ADC). Additionally, buffer op-amp circuits shown in Figure5 interface between the voltage dividers and the Arduino to reduce the error from the source impedance of the flex sensors.

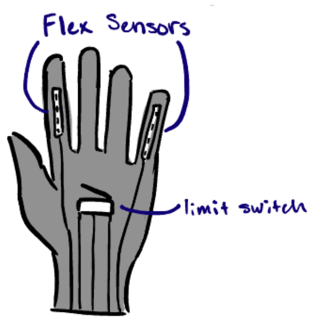

Figure 4: Detailed drawing of the player glove and its sensors

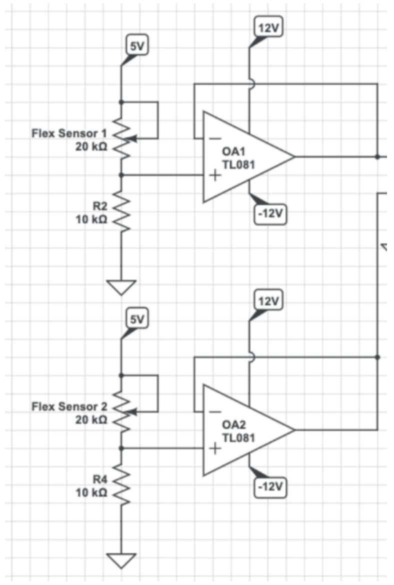

Figure 5: Dual flex sensor voltage dividers with input buffers for signal processing

2.3.3 LCD Display

After both the robotic hand and the player have made their moves, the Arduino proceeds to determine the winner and showcases the score on the LCD Display, which is linked via I2C. Whenever a winner is identified, their score is incremented by one and presented on the display, as depicted in Figure 6. Additionally, the full Arduino program flowchart is detailed in Figure 7.

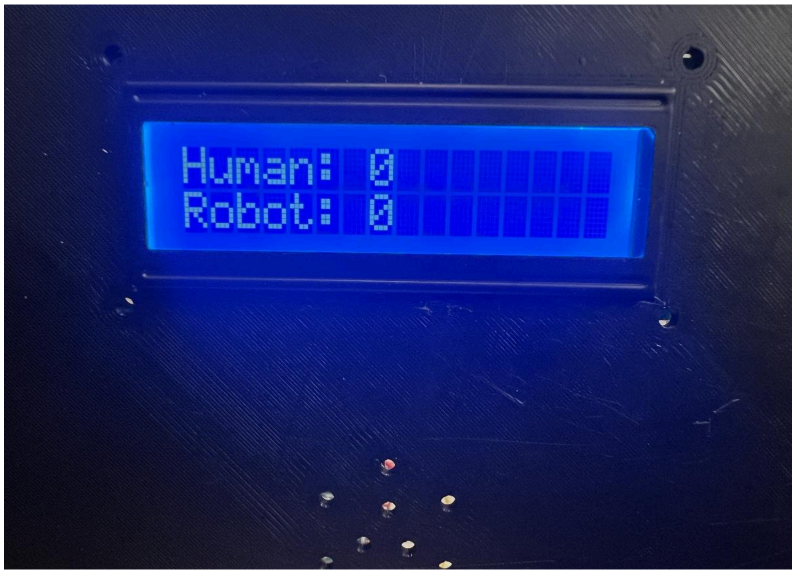

Figure 6: LCD display

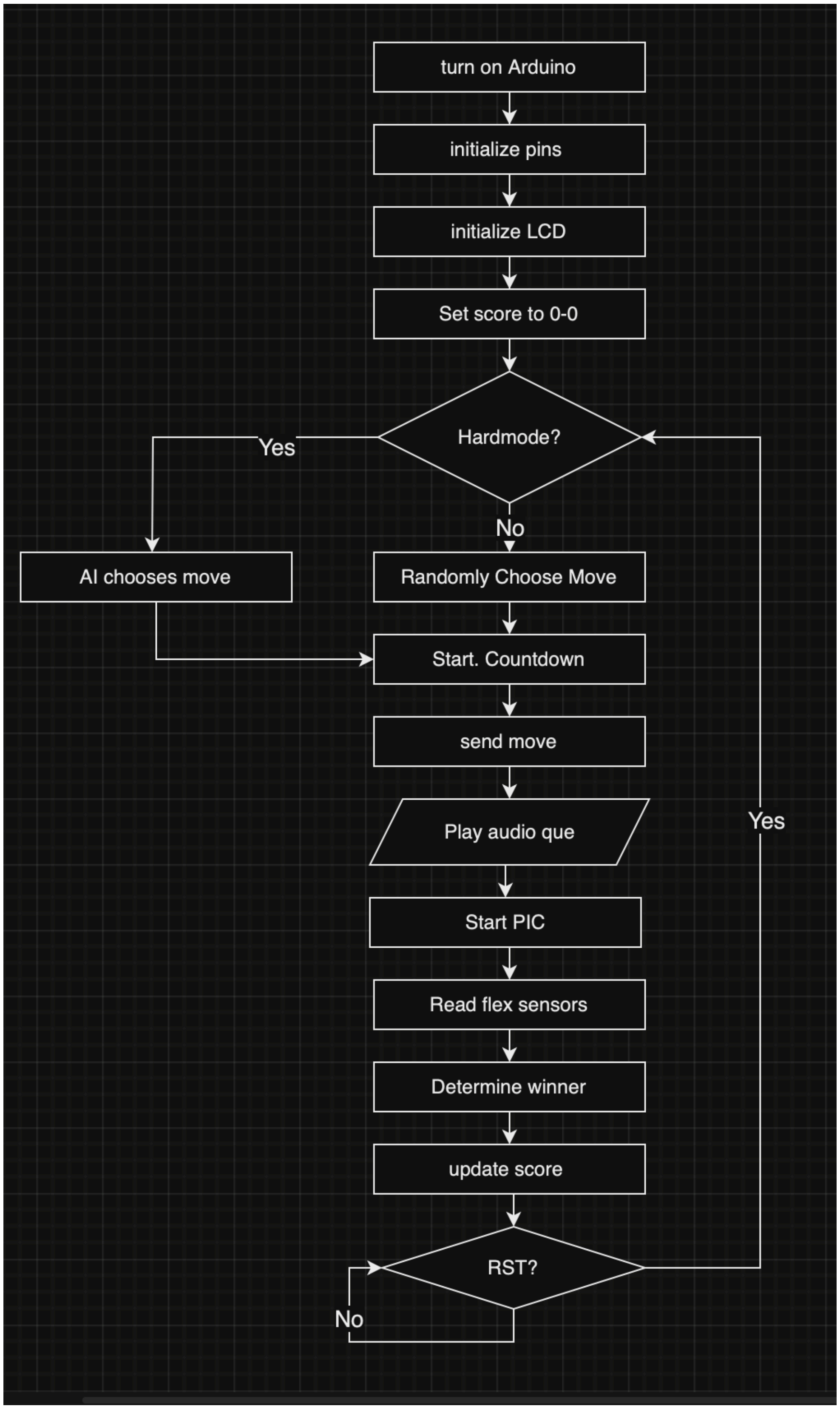

Figure 7: Arduino Program Flowchart

3 Design Evaluation

3.1 Output Display Device

The output display device that the system utilizes is a liquid crystal display (LCD) which the Arduino microcontroller communicates with using I2C and displays the current score of the player and the system. Once the system determines the winner of each round, either the player or the system itself, the system increases the score of the winner and then updates their corresponding score onto the LCD for the player to see. The display meets all requirements and has never malfunctioned, 10/10.

3.2 Audio Output Device

The system initially worked using the Icstation Recordable Sound Module, however, the module was burnt out while transferring to a solder board. There is a recording of the system working together on a solderless breadboard with this component, but the final demonstration ended up utilizing a Dfrobot Speaker V2.0. This speaker is a digital buzzer, so instead of playing an audio file, the speaker was programmed to play four frequencies with pauses to symbolize the "rock, paper, scissors, shoot!" audio. This reliably achieved the intended purpose of telling the player when to throw their move. An unfortunate side effect, however, was that the delay on the speaker output occasionally caused the PIC not to receive the move from the Arduino which was not the case with the original sound module. Considering the reliability and the research that was applied to find a speaker that could play an mp3 file as well as the subsequent modifications made to the device, we think that a score of 5/10 is reasonable for this section.

3.3 Manual Data Input

The manual data input to the system consists of a difficulty switch that tells the system to play in either hard mode or easy mode and a ready button which the player presses to indicate they are ready to start a round against the system. Additionally, there is a power switch that turns on the device. These inputs all functioned correctly and reliably, but based on the little research required to implement them, a score of 8/10 is suitable for this section.

3.4 Automatic Flex Sensor

The reliability of the flex sensors can be measured by how often the system reads the correct move. Individuals with large hands are able to make more substantial bends in the sensor. The system has accurately detected all moves thrown by men over a sample of three individuals and over fifty moves. However, some women tend to have difficulty throwing scissors because the bottom sensor is not bent enough to produce a noticeable difference. Since there is not a feasible way to increase the accuracy of the sensor and it works reliably for most individuals, this section should have a 9/10.

3.5 Actuators, Mechanisms and Hardware

The robotic hand and servo motors worked together to actualize the digital rock-paper-scissors move as a 3D gesture. The hand started in a “rock” position with all fingers closed and would open fingers from there, using the servo motors, to mimic the human hand signs. It would then return to the “rock” position to signal the end of the round.

Although the different moves were distinguishable, there were a couple of issues with the servos. Our group was most familiar with continuous servos, so we chose to use those to operate the fingers. We later realized that continuous servos do not hold a position after being stopped, which meant that the robotic fingers would immediately relax after a move was played. This made differentiating between scissors and rock more difficult if the player did not see the robot moving. By the time we realized this, we decided that we may not finish in time if we chose to change to positional servos since they would require completely different mounting to the stage and hand as well as a different program for the PIC.

Although the servo choice could have been better, a lot of time and effort was put into designing the robotic hand and getting it to work with the servo. In the end, the hand operated fundamentally as expected and required extensive independent research, so we think that a score of 8.5/10 for this section is reasonable.

3.6 Logic, Processing, and Control

To operate the servos, we came across a PIC that could output two PWM signals (CCP1 and CCP2). While we managed to rotate the servos quickly and accurately in the end, grasping the hardware PWM logic and code posed quite a challenge. However, our success in implementing a reliable PWM system also meant a significant improvement in our group's understanding of PWM.

Initially, we had planned to control the speaker with the PIC as well, but that turned out to be extremely difficult with little success. Consequently, we decided to move the speaker to the Arduino. Additionally, dealing with the PIC pins that received signals from the Arduino proved to be extremely challenging, as they often failed to read the signals properly. These two factors led us to rate the PIC component 6/10 in terms of performance.

On the other hand, the Arduino seamlessly handled each task with minimal debugging required. Once we shifted the speaker to the Arduino, we were able to achieve a proper response from the speaker in less than an hour. Furthermore, the Arduino posed no issues with the LCD display or the flex sensors. Therefore, we give the Arduino component a perfect score of 10/10.

4 Bill of Materials

Table1. List and cost of all materials in the system

Part/Component	Price ($)	Source
Flex Sensor x2	25.66	Spectraflex
Mark Forge Carbon Black Filament	20.99	Makerspace
Bamboo Black PLA Filament	5.23	Makerspace
Tower Pro Micro Servo Motor x2	3.99	Makerspace
SunFounder IIC I2C TWI 1602 Serial LCD	9.99	Makerspace
Arduino Uno	30.00	Makerspace
PIC 16F886	8.80	Makerspace
Permanently open Push Button	0.90	Makerspace
2-way Switch	0.50	Makerspace
MDF 1'x2' Sheet	30.00	Makerspace
LM741 op amp x3	2.61	Makerspace

5 Lessons Learned

5.1 Parts Selection

A significant lesson learned was the necessity of conducting thorough research on components. In this project's context, it became evident that more extensive research should have preceded the selection of motors to control the hand. Despite opting for continuous servo motors based on familiarity, the challenge arose in utilizing the PIC to transmit PWM signals. These signals were not only required to direct the motors accurately but also to achieve specific distances and speeds, which proved to be quite challenging. Consequently, the process of coding and fine-tuning the motors spanned nearly two weeks.

5.2 Modular Design Management

While a modular design approach was adopted overall, it was not executed efficiently. The intention behind identifying modules was to simplify the understanding and operation of sensors like flex sensors and actuators such as motors. The idea was that by starting with simple functionalities, the group could gradually build towards more complex tasks with ease. However, this approach did not yield the expected results. When attempting to integrate all components into the project simultaneously, debugging became challenging, leading to the eventual segmentation of the code into modules.

Moreover, the deficiency in proper modular design resurfaced when transitioning the project to a solder board. Instead of assembling the board incrementally, we attempted to complete it in one go, resulting in numerous shorts and issues that were nearly impossible to debug.

5.3 Take advantage of debugging and testing methods

In Design 5, we were introduced to microcontroller programming for the first time. However, debugging tools like LEDs, multimeters, and oscilloscopes were seldom utilized. Instead, the primary approach involved rewriting the code and adjusting minor details in the hope of resolving issues. However, this method proved inadequate for Mechatronics. Debugging servos, switches, and various other components necessitated the use of all available lab devices and debugging methods.

6 References

[1] “Amazon.com: ICSTATION Recordable Sound Module, button control sound chip 8m MP3 WAV Music Voice Player Programmable Board with speaker for DIY Birthday music box greeting card mother’s day : Electronics,” Amazon, https://www.amazon.com/Icstation-Recordable-Button-Music-Board/dp/B01M35VHY5 (accessed May 1, 2024).

A Appendix

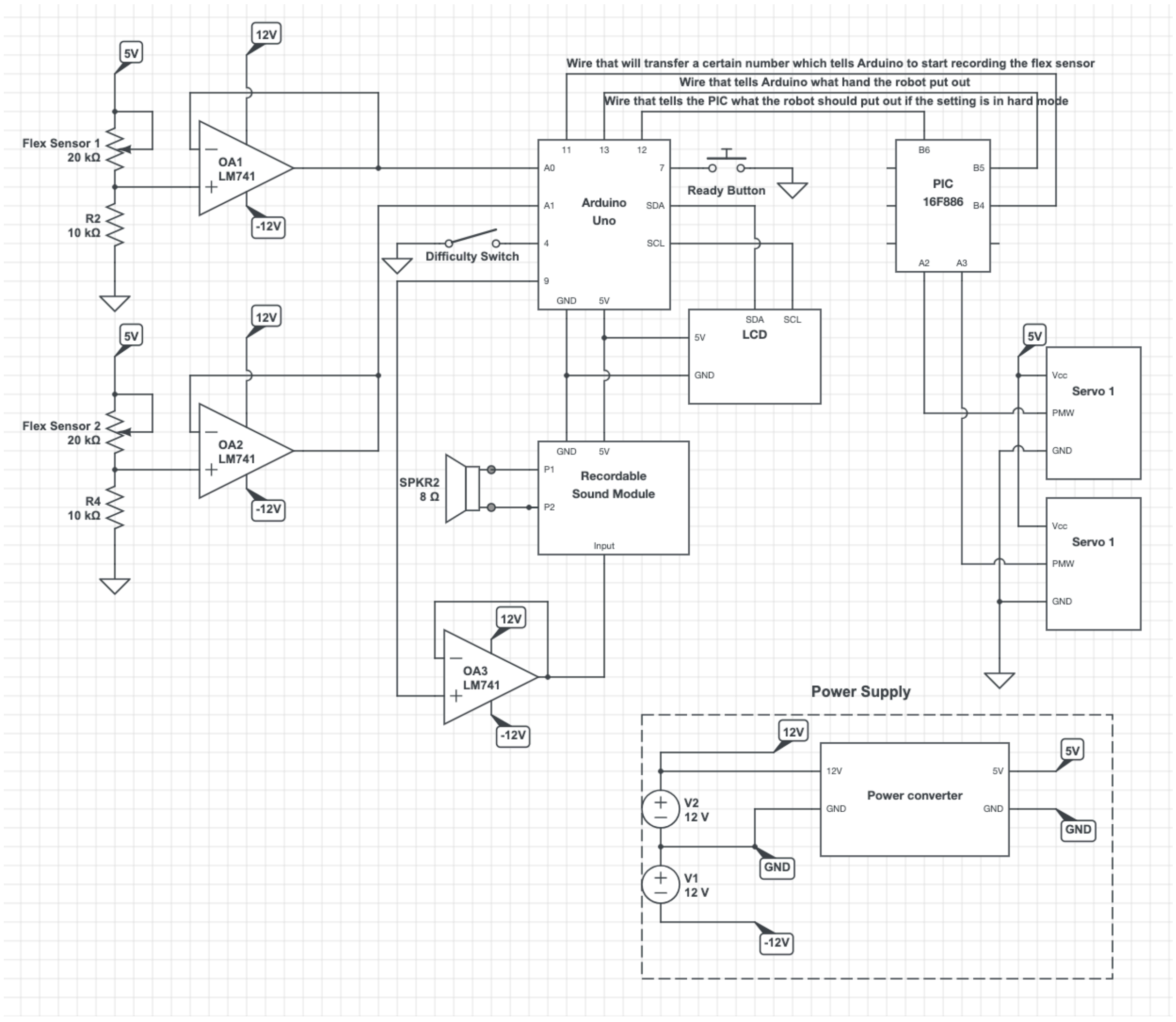

Figure A1. Full Circuit Schematic

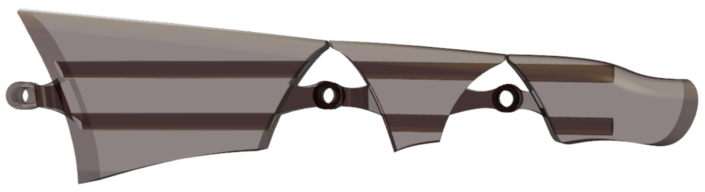

Figure A2. Extended finger - top wire pulled

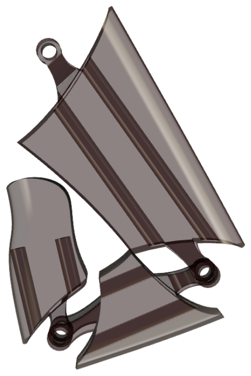

Figure A3. Contracted finger - bottom wire pulled

B Appendix - Arduino code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x20, 16, 2);

int flex1 = A0; // first flex sensor
int flex2 = A1; // first flex sensor
int RST = 7; //limit switch on glove
int hardMode = 4; //hardmode switch
int speaker=9;

int PIC_STRT = 13;
int PIC_MV1 = 12;
int PIC_MV2 = 11;
int scR;
int scH;
bool whoWon; //true when the robot wins
int lastRobo = 3;
int lastPlaya;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(flex1, INPUT);
  pinMode(flex2, INPUT);
  pinMode(hardMode, INPUT_PULLUP);
  pinMode(RST, INPUT_PULLUP);

  pinMode(PIC_STRT, OUTPUT);
  pinMode(PIC_MV1, OUTPUT);
  pinMode(PIC_MV2, OUTPUT);

  digitalWrite(PIC_STRT,LOW);
  digitalWrite(PIC_MV1,LOW);
   digitalWrite(PIC_MV2,LOW);
  digitalWrite(speaker,HIGH);

  scR =0;
  scH = 0; //set human and robot scores

  lcd.init();
  lcd.backlight();
  dispScore();
}

void loop() {
  // put your main code here, to run repeatedly:
  while(digitalRead(RST)){
    delay(250);
  }
  //digitalWrite(speaker,LOW);
  Serial.println("speaker start");
   tone(speaker, 200, 500);
   delay(200);
  tone(speaker, 200, 500);
  delay(200);
  tone(speaker, 200, 500);
  delay(200);
  tone(speaker, 300, 600);
  int robot_mv = sendMove(PIC_MV1,PIC_MV2, hardMode); //send the robot move to the PIC
    delay(200);
  digitalWrite(PIC_STRT,HIGH);  //starts the PIc subroutine
  Serial.print("speaker stop");
  //noTone(speaker);
  delay(500);
  getWinner(robot_mv, getPlayerMV()); //read player move, determine winner, and update score
  delay(250);
  dispScore(); //display new score
  digitalWrite(PIC_STRT,LOW);
  digitalWrite(PIC_MV1,LOW);
    digitalWrite(PIC_MV2,LOW);
  delay(200);

}

bool readFlex(int flex){
  int val = analogRead(flex); //read flex sensor
  Serial.println(val);
  if (val <220){ return true;}
  else{ return false; }
}

void dispScore(){
  lcd.setCursor(0,0);
  lcd.print("H");
  lcd.print("u");
  lcd.print("m");
  lcd.print("a");
  lcd.print("n");
  lcd.print(":");
  lcd.setCursor(7,0);
  lcd.print(scH);
    lcd.setCursor(0,1);
  lcd.print("R");
  lcd.print("o");
  lcd.print("b");
  lcd.print("o");
  lcd.print("t");
  lcd.print(":");
  lcd.setCursor(7,1);
  lcd.print(scR);
}

int pickMove(int hard){
  //bool isHard = digitalRead(hard);
  if(digitalRead(hard) == HIGH){
    if (lastRobo == 3){
      lastRobo = random(3);
      return lastRobo;
    } else if (lastRobo != 3 && whoWon == true){
      return lastRobo;
    } else{
      if (lastPlaya == lastRobo){
        lastRobo = random(3);
      return lastRobo;
      } else if (lastPlaya == 0 && lastRobo == 1){
        lastRobo = 2;
      return lastRobo;
      }else if (lastPlaya == 1 && lastRobo == 2){
        lastRobo = 0;
      return lastRobo;
    } else if  (lastPlaya == 2 && lastRobo == 0){
        lastRobo = 1;
      return lastRobo; }
    }
  } else if (digitalRead(hard) == LOW){
    lastRobo = random(3);
    return lastRobo;
  }
```

```
}


int sendMove(int pin1,int pin2, int hardMode){
  int move = pickMove(hardMode);
  switch(move){
    case 0: //rock
      digitalWrite(pin1, HIGH);
      digitalWrite(pin2, LOW);
      Serial.println("rock");break;
    case 1: //scissors
      digitalWrite(pin1, LOW);
      digitalWrite(pin2, HIGH);
      Serial.println("scissors");break;
    case 2: //paper
      digitalWrite(pin1, HIGH);
      digitalWrite(pin2, HIGH);
      Serial.println("paper");break;
    default: //rock
      digitalWrite(pin1, HIGH);
      digitalWrite(pin2, LOW); break;
  }
  return move;
}

int getPlayerMV(){
  bool f1 = readFlex(flex1);
  bool f2 = readFlex(flex2);
  if(f1 && f2){
  lastPlaya = 0;
  return 0; } //rock
  else if(!f1 && !f2){
  lastPlaya = 2;
  return 2; } //paper
  else if(!f1 && f2){
  lastPlaya = 1;
  return 1;} //scissors
  else{ return 0;
  lastPlaya = 0;} //default is rock
}

void getWinner(int r,int h){
  if(r == h ){return;} // tie
  switch(r){
    case 0:         //robot throws rock
    if(h == 1){
      scR = scR +1;
      whoWon = true;}    //human throws scissors and loses
    else{scH = scH +1;
    whoWon = false;};break; //human throws paper and wins
    case 1:         //robot throws scissors
    if(h == 2){scR = scR +1;
    whoWon = true;}    //human throws paper and loses
    else{scH = scH +1;
    whoWon = false;};break; //human throws rock and wins
    case 2:         //robot throws paper
    if(h == 0){scR = scR +1;
    whoWon = true;}    //human throws rock and loses
    else{scH = scH +1;
    whoWon = false;};break; //human throws scissors and wins
  }
}
```

C Appendix - PIC code

```
 #CONFIG
    __config _CONFIG1, _INTRC_OSC_NOCLKOUT & _WDT_ON & _MCLRE_ON &
_LVP_OFF & _CP_OFF
#endconfig
define OSC 4
OSCCON = %00010010

ansel = 0

duty var word
duty2 var word
strt var PORTA.1
mv1 var PORTA.2
mv2 var PORTA.3

'rec_mv_1 VAR PORTB.6
'rec_mv_2 var PORTB.5
'rec_mv_3 var PORTB.4

TRISC.1 = 0
TRISC.2 = 0
TRISB = 0
TRISA = 1

CCP1CON = %00001100
CCP2CON = %00001100
T2CON = %00000110

PR2 = 155

duty = 1
duty2 = 1

'low rec_mv_1
'low rec_mv_2
'low rec_mv_3
pause 180


mainloop:
   high PORTB.3
```

```
    WHILE(!STRT)
        PAUSE 1
    WEND

    low PORTB.3

    if (mv1 and !mv2) then
        gosub ClosePap
        pause 60
    elseif (!mv1 and mv2) then
        gosub Scissors
        pause 60
        gosub CloseSic
    elseif (mv1 and mv2) then
        gosub Paper
        pause 60
        goSub ClosePap
    endif

    pause 80



goto mainloop

Scissors:

    duty = 1
    duty2 = 25
    CCP1CON.4 = duty.0
    CCP1CON.5 = duty.1
    CCPR1L = duty >> 2
    CCP2CON.4 = duty2.0
    CCP2CON.5 = duty2.1
    CCPR2L = duty2 >> 2
    pause 10
    duty = 0
    duty2 = 0
    CCP2CON.4 = duty2.0
    CCP2CON.5 = duty2.1
    CCPR2L = duty2 >> 2
    CCP1CON.4 = duty.0
    CCP1CON.5 = duty.1
    CCPR1L = duty >> 2
    return

Paper:

    duty = 1
    duty2 = 1
    CCP2CON.4 = duty2.0
    CCP2CON.5 = duty2.1
    CCPR2L = duty2 >> 2
    CCP1CON.4 = duty.0
    CCP1CON.5 = duty.1
    CCPR1L = duty >> 2
    pause 10
    duty = 0
    duty2 = 0
    CCP2CON.4 = duty2.0
    CCP2CON.5 = duty2.1
    CCPR2L = duty2 >> 2
    CCP1CON.4 = duty.0
    CCP1CON.5 = duty.1
    CCPR1L = duty >> 2
return

ClosePap:

    duty = 25
    duty2 = 25
    CCP1CON.4 = duty.0
    CCP1CON.5 = duty.1
    CCPR1L = duty >> 2
    CCP2CON.4 = duty2.0
    CCP2CON.5 = duty2.1
    CCPR2L = duty2 >> 2
    pause 10
    duty = 0
    duty2 = 0
    CCP1CON.4 = duty.0
    CCP1CON.5 = duty.1
    CCPR1L = duty >> 2
    CCP2CON.4 = duty2.0
    CCP2CON.5 = duty2.1
    CCPR2L = duty2 >> 2

return
```

```
CloseSic:
   duty = 25
   CCP1CON.4 = duty.0
   CCP1CON.5 = duty.1
   CCPR1L = duty >> 2
   pause 10
   duty = 0
   CCP1CON.4 = duty.0
   CCP1CON.5 = duty.1
   CCPR1L = duty >> 2
return
```