Computer Science Final Project

Yuvraj Singh

December 11th 2024

Index

Section 1:

1.1 Identifying Connections:

- CSE1120: Structured Programming 2
- CSE1210: Client-Side Scripting 1

1.2 Explain Outcome Integration:

CSE1120: Structured Programming 2:

1. Use of IPO structure in the code that will be controlling the robotic arm
2. Implementation of selection and iteration control structures n Arduino code.
3. Use of modular blocks (functions) to control each of the servos and read sensor inputs.
4. Demonstration of algorithmic idioms like checking maximum/minimum rotational angles for safety.
5. Debugging and revision of code for optimal motor response.

CSE1210: Client-Side Scripting 1:

1. Dvelopment of a website using HTML to showcase the project.
2. Use of the language to structure and display content, including project explanation, images, and progress
3. Website design follows principles of an organized layout, with appropriate headings, sections, and formatting for user readability.
4. Demonstrates understanding of the client-side structure of the web using HTML elements such as paragraphs, lists, images, and links.

This project integrates programming and web development skills. The code for the robotic arm is written using a structured programming approach, including control structures, arm design, and debugging. These practices are directly aligned with CSE1120 outcomes and ensure that the device responds predictably and reliably to sensor inputs from the rotational sensor.

To present the project, a website is created using HTML, fulfilling the outcomes of CSE1210. The website clearly documents the project's purpose, components used, how the code works, and the results, making it accessible to others. By combining these skills, the project not only demonstrates technical ability in programming but also effective communication through a well-organized web page.

Section 2:

2.1 Project Proposal:

In many laboratory environments, handling hazardous acids and chemicals poses significant risks to human health and safety. Current manual methods of transferring these substances increase the likelihood of spills, exposure, and long-term health effects for lab personnel. To mitigate these dangers, I will be constructing a Robotic Arm that will be capable in picking up and moving a 30 mL beaker glass

Deliverables:

- Arduino IDE
- Processing
- Website on page number:

In a variety of scientific settings, handling dangerous acids and chemicals poses serious dangers to people's health and safety. Because these products are being handled by hand, there is a greater chance of spills, exposure, and long-term health consequences for lab staff. A system that is precise, affordable, and remotely operated is required to lessen these risks. In order to minimize direct human touch and improve overall lab safety, this project intends to develop and construct a robotic arm using Arduino technology that can precisely and safely handle hazardous chemicals in a laboratory setting.

2.2 Timeline and Milestones:

Week 1: Finalize on project idea and plan

Week 2: Work on flowchart and begin Website portfolio

Week 3: List out and research materials and extra electronics needed for the project
work on website (Add style sheet, home page, about, etc)

Week 4: Begin Robotic arm model using TinkerCAD and Fusion360 and also begin code for each servo motor

Week 5: Print and test model, Add to website for fails and successes. Continue documenting progress on the HTML webpage

Week 6: Continue testing for both model and code

Week 7: Finalizing and polishing website

Week 8: Final testing and fixes if needed

2.3 Terminology, Tools, and Resources:

- Arduino IDE:

The Arduino IDE (Integrated Development Environment) is used to create code for its microcontroller, the program is a simplified version of C++. Due to its simple and easy to use interface, this makes it perfect for beginners and also enthusiasts

The IDE is equipped with a code editor, toolbar, and serial monitor. Its primary functions are real-time error checking, automated formatting, and syntax highlighting. Users can access pre-installed sample code, add external libraries, and monitor sensor data via the serial connection.

- Processing:

Processing is an open-source programming language created for making visual and interactive applications, it allows users to easily create graphics, animations, simulations, and interactive programs using simple code.

Processing is based on Java but uses a simplified syntax, making it beginner-friendly and ideal for creative coding. It includes a built-in editor and supports real-time visuals, making it popular in fields like digital art, data visualization, and educational projects. Processing can also communicate with external hardware like Arduino, enabling interactive computer projects.

Resources and Cost:

Sources I used:

- Mastering Servo Control: PCA9685
- Complete guide to PCA9685
- Mastering Servo Control w/ PCA9685 and Arduino

Items Bought:

- MG90S Servo (4): 20.99
- MG995 Servo (1): 11.49
- PCA9685 Servo Motor Driver (1): 10.59
- 5 Volt 2 Amp Power Adapter (1): 11.92

Section 3:

3.1 Decomposition of the Project:

- Input: Input will be taken manually, depending on the users desired angle through sliders on Processing
- Processing: Takes the input and maps it to a pulse for the servo driver to read and understand
- Output: Turns servo to its angle

Flowchart:

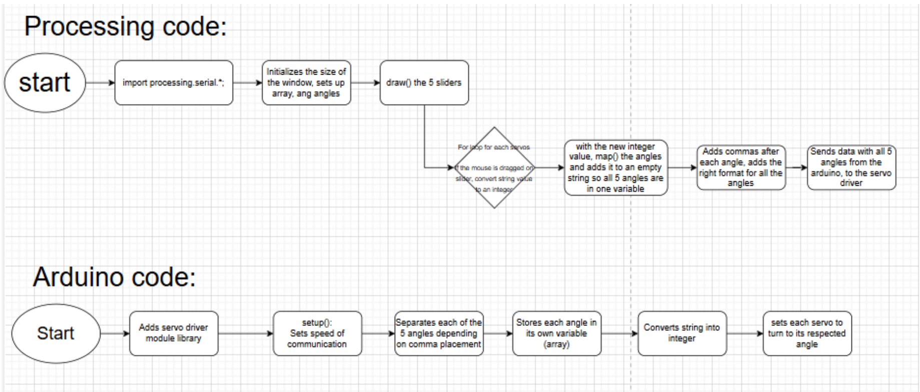
Processing code:
start
import processing.serial.*;
Initializes the size of the window, sets up array, ang angles
draw() the 5 sliders
For loop for each servos
If the mouse is dragged on slider, convert string value to an integer
with the new integer value, map() the angles and adds it to an empty string so all 5 angles are in one variable
Adds commas after each angle, adds the right format for all the angles
Sends data with all 5 angles from the arduino, to the servo driver
Arduino code:
Start
Adds servo driver module library
setup(): Sets speed of communication
Separates each of the 5 angles depending on comma placement
Stores each angle in its own variable (array)
Converts string into integer
sets each servo to turn to its respected angle

Figure 1: Flowchart for Processing to Arduino to Servo Driver to Servos

3.2 Website Design:

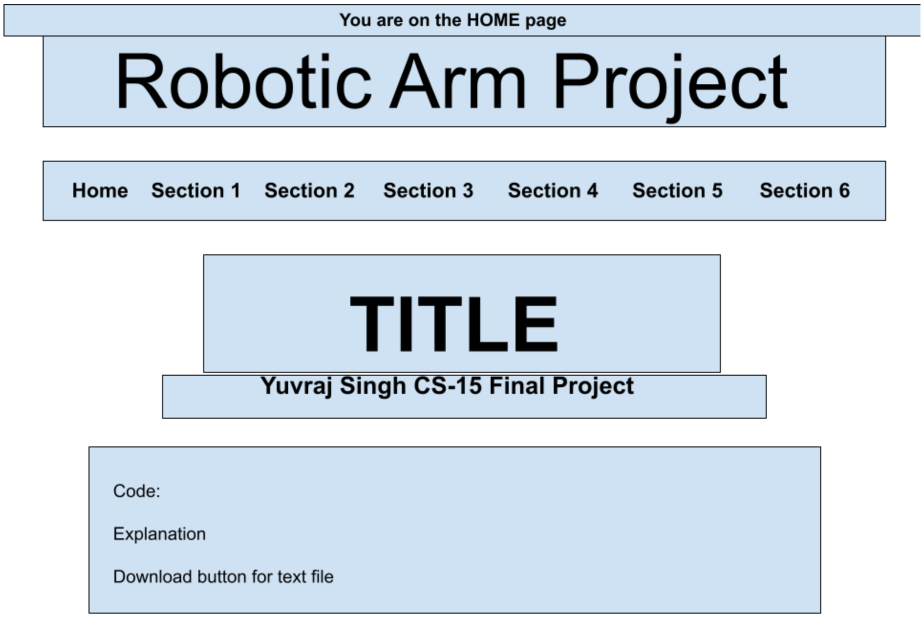

Figure 2: Home Page Wireframe

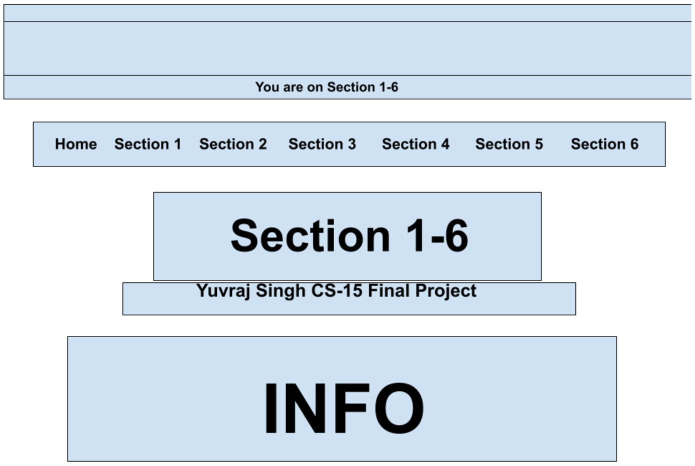

Figure 3: Section 1-6 Page Wireframe

3.3 Integration of Skills:

From CSE1120, I used many skills related to structured programming. For example, I used loops, if statements, arrays, and functions to create a clear structure in both my Processing and Arduino code. My program follows the Input–Processing–Output (IPO) model: user inputs are taken through sliders in Processing, which are then mapped to servo angles and sent through the serial port to the Arduino. The Arduino receives the data, processes each angle, and moves the servos to the right position. I also had to debug and test my code several times to make sure the servos responded properly and didn't go over their limits. This showed my understanding of decision-making, repetition, and how to break problems into smaller steps.

From CSE1210, I used my skills in HTML and CSS to design a website that explains what my project is, how it works, and the challenges I faced. I organized the site into sections like Project Description, Code Snippets, Flowchart, and Screenshots. These match the outcome of showing how a website can present information clearly to an audience. I also learned how important client-side scripting is for sharing work and showing how it connects to real-world applications, like robotics and engineering.

Section 4:

4.1 Modules:

Input: The input part of the code was the Processing code. Through the javascript UI window, the user can slide any slider corresponding to the servo they want to turn. This is the input aspect of the project, as this is recieving input from the user itself.

Processing: This part is when the Arduino converts the given input angles into a pulse which can be read by the servo driver module. It then Converts it to a integer value which turns the corresponding servo

Output: Servo is turned to the desired angle

4.2 Website:

Linked on Teams.

Section 5:

5.1 Testing and Debugging:

Each module works to its desired intent, The input takes in the value of the slider and converts it to the servos flawlessly. Problems that I did face was the confusion between conditions, when it came to for loops, I was muddled in figuring out how to make it work. I knew I needed to use for loops so that my code can run the same thing for each of the 5 servo motors. Eventually I had figured it out from the use of array values.

5.2 Project Evaluation:

From the original project, I had changed the type of servos. Instead of using the plastic gear SG90 servo motor, I had upgraded to a metal gear servo motor and also a stronger base servo to hold and maintain weight from the metal servos. Because of the advanced servos and also the high torque that comes with it, power was a major issue. I had added a servo driver module to tackle and resolve that problem. Previously I used a breadboard to control my servos with potentiometers for each servo, I had changed the control to a Javascript UI interface instead. This had resulted in a more smoother control and also fixed the issue of my jittery servo motors form my previous model.

Future Improvements:

3D print the model for the robotic arm, add a gear system to the claw to make it open and close. Include a text box to enter your own angle value, the servo will turn to that angle that you put in. Add an ESP32 Bluetooth module to eliminate the use of a wire to my laptop for transmitting the data.

6. Code:

Processing code:

```
import processing.serial.*;
//allows to send data to the arduino board
//this is needed so that the data from my laptop can go into
     the Arduino board
Serial dataPort;
//Creates a variable that will store serial data
//Serial is needed for me to communicate with the Arduino
    board by a serial connection. In this case it is the wire
     to the Arduino and my laptop.
int[] servoANGLES = {90, 90, 90, 90, 90};
// Inital angle of the servos, once the code starts, the 5
    servos turn 90 degrees.
int numServos = 5;
//This variable is setting the number of servos im using,
    which will also be used for the number of sliders
int sliderWidth = 180;
//the width of each slider in px (pixels) that goes from
    left to right
int sliderHeight = 20;
//the height of each slider in px (pixels) that goes from
    top to bottom

void setup() {
//Setup
  size(500, 250);
  //dimensions of the window in px, 500px wide and 250px tall.
  printArray(Serial.list());
  //This prints out which port my Arduino is connected to. In this case, it is COM3, so it will always print COM3 in the console. This is needed so that all the data can be transferred correctly thorugh the right port.
  dataPort = new Serial(this, "COM3", 9600);
  //The serial port variable "dataPort: will store the serial connection. this will be used to send data to and from the laptop and Arudino.
  // new Serial() creates a new serial object that controls the communication and port, the "this" is reffering to this Processing document, as it will be sending data from this document to the arduino. The "COM3" sets the port in which data will send, and then the "9600" is the baud rate, which is the speed of communication for the Arduino. As this must match with my arduino code which is also 9600 baud.

}

void draw() {
//This is the drawing for the interface, for both the sliders and also the colors
  background(0);
  //Black background
  fill(255);
  //title text color is set to white
  textSize(14);
  //Font size is set to 14px (pixels)
  text("Robotic␣Arm␣for␣Laboratory␣Purpose", 1, 20);
  //draws and displays the text and also sets the x and y position
  text("Yuvraj␣Singh␣CS-15␣Final␣Project", 305, 20);
  //draws and displays the text and also sets the x and y position


  for (int ServoNum_X = 0; ServoNum_X < numServos; ServoNum_X++) {
   //A for loop is a loop that is ran a certain number of times
   //In this case, it is running until 5, I set the servonum as zero for the loop to use that value to know when to stop and to also know which servo is being used.
   //the < numServos is bascially <5, as the numServos value is declared as 5 at the beggining. servonum++ is for when everytime the for loop is completed, the servonum value is increased by +1
```

```
    int y = 60 + ServoNum_X * 35;
    // this line is for the position of each slider on the y
       -axis, each slider will be 35 pixels apart from each
       other in a equal distance.

    //How it is displayed:
    //when servonum = 0: y = 60 + 0 * 35
    //y = 60 (The first slider will be displayed at y = 60px
       )

    //when servonum = 1: y = 60 + 1 * 35
    //y = 95 (The second slider will be displayed at y = 95
       px)

    //when servonum = 2: y = 60 + 2 * 35
    //y = 130 (The third slider will be displayed at y = 130
       px)

    //when servonum = 3: y = 60 + 3 * 35
    //y = 165 (The fourth slider will be displayed at y =
       165px)

    //when servonum = 4: y = 60 + 4 * 35
    //y = 200 (The fifth slider will be displayed at y = 200
       px)



    fill(180);
    //grey background for slider
    rect(50, y, sliderWidth, sliderHeight);
    //creates a rectangle with x and y values given, the y
       value is the ones that we saw above, it then uses
       dimensions that were declared at the beginning of the
        code (sliderWidth = 180px & sliderHeight= 20px)

    int sliderPos = (int)map(servoANGLES[ServoNum_X], 0,
       180, 0, sliderWidth);
    //this is ths position of the slider that will move, the
        map() function is formatted like this: map(value,
       fromLow, fromHigh, toLow, toHigh)
    // the value is the angle of one servo, the fromlow part
        is your lowest value that your servo motor can
       achieve; which is zero, fromHigh is your highest
       value from the angle which is 180.
    //toLow is the lowest value you want to/ convert to, in
       this line, as we are drawing the length of the
       rectangle for each time it is drawn, our lowest will
       be zero.
    //toHigh is your value that is your highest, which is
```

```
the sliderWidth, as we do not want the blue rectangle to be drawn off the slider.

    fill(5, 5, 100);
    //makes slider blue
    rect(50, y, sliderPos, sliderHeight);
    //creates the recatngle to show how "filled" the other slider is, this is the rectangle that will be moved to determine the angles.





    fill(255);
    //text color is white
    text(servoANGLES[ServoNum_X] + "  ", 240, y + 15);
    //draws a text of the current servo angle on the right side of each slider, it is then added with a degree sign and also has the respected x and y coordinates, the x values are the same but the y value is added by +15 because then the angles would not be beside the sliders


    //This is ALL under a for loop, once one lap is completed, the servonum variable is added by +1 and it all repeats again until the servonum is < 5

}

  fill(255);
  //this is for labelling each slider, as i set the text color to white
  textAlign(LEFT);
  //aligns the text to the left side

 //These are labels for each of the 5 servo motors and their desired x and y coordinates on the window
  text("BASE", 1, 75);
  text("Joint␣#1", 1, 110);
  text("Joint␣#2", 1, 145);
  text("Joint␣#3", 1, 180);
  text("Joint␣#4", 1, 215);


  Data();
  //Calls a function. The use for it is stated below
}

```

```
114 void mouseDragged() {
115 //mouseDragged() is a built in function that runs a code whenever you hold down the mouse button and drag/move the mouse
116   for (int ServoNum_Y = 0; ServoNum_Y < numServos; ServoNum_Y++) {
117   //Another for loop for repeating the code 5 times for each servo
118     int y = 60 + ServoNum_Y * 35;
119     //setting the y value, this is the same line of code in line 46. This is repeated so that I can use the y value and see if the mouse is in the rectangular bar.
120
121     //An if statement that is for when the mouse is in the dimensions of the sliders and when they are dragged
122     if (mouseY > y &&
123     //First condition that needs to be met: If the y coordinate of the mouse on the window is greater than the "y", the "y" value will be different each and every time because of the code in line 118. This part of the if statement is saying if the y value of the mouse cursor is greater, it is underneath the slider. The greater your y value is, the more lower you are on the screen.
124         mouseY < y + sliderHeight &&
125         //This next condition is for when it needs to be kept between the slider, it is saying that the y value for the mouse is the "y" plus the height of the slider. the "y" value for the rectangles are for the top left corner of the rectangle. To get the entire distance from the top of the window to the bottom part of the slider. So far, we have stated that the cursor needs to be between the sliders for it to be true.
126         mouseX > 50 &&
127         //The next condition is when the x value for the mouse cursor is greater than 50. If you plot the sliders on a xy plane, you will see that the left corner is at 50. This was a good way to visualize and understand about the exact placement I wanted the mouse cursor to be, the only thing is that the y axis needs to be going down, as the higher y-value, the lower you go on the window.
128         //The x value for the mouse cursor needs to be greater than 50, basically meaning that it needs to be above 50, this isnt what we want but we will add our next limit in the following line.
129         mouseX < 50 + sliderWidth) {
130         //This is the final limit for the x values, as I did
```

```
           the same thing I did for the height of the slider. I did 50, as that is the left corners of the slider plus the entire slider width/length. This value basically gave the value of the right corner of the slider. As then I set the next condition to be that the mouse x value needs to be lower than 50 + the slider width

          //This entire if statement is basically setting the dimensions of the sliders and delaring them that if the mouse is dragged in there, do the following

        int mapped_angle = (int)map(mouseX, 50, 50 + sliderWidth, 0, 180);
        //the map() angle has the format like this: map(value, fromLow, fromHigh, toLow, toHigh)
        //You can see the same format in the map() function I used, as the value that i am using will be the mouse x value for where the mouse is/how far the slider got dragged, the fromLow is the lowest value that you origionally used which is the beginning of the slider (50), and then the highest value from the origional value (50+ Slider Width = very end of slider)
        //It then maps those given values to your new value, the toLow is the lowest value in your new range which is 0 degrees, and then the toHigh is the highest value in your range which is 180.
        //For the map() function, it is basically a ratio. For instance, the x value for the left side of the slider is 50, and that is in ratio to 0 degrees. And then the right side of the slider is 50 plus the sliderwidth which is in ratio to 180
        //With the ratio, it converts the mouse x value to an apropriate degree, that is set as an integer
        servoANGLES[ServoNum_Y] = constrain(mapped_angle, 0, 180);
        //This sets the angle of 1 servo depending on the servo number, constrain keeps the angle between the lowest and the highest value which was done in this line. As my constraints are 0 degrees to 180 degrees, The mapped_value is the value that is being kept in the range of the set values of low to high.
        //As the ServoNum_Y value keeps changing, the array will set its new angle as its own variable.
        //For example, if the angles are (90,45,60,120,180).
        //the ServoNum_Y will keep changing value and setting each number as its own angle. It would look like
```

```
     this:
   //servoANGLES[0]=90
   //servoANGLES[1]=45
   //servoANGLES[2]=60
   //servoANGLES[3]=120
   //servoANGLES[4]=180



  }
 }
}


void Data() {
//function
 String data = "";
 //empty string that will store data for all 5 angles for
   each servo
 for (int ServoNum_Z = 0; ServoNum_Z < numServos;
   ServoNum_Z++) {
 //Another for loop for each of the 5 servos
  data += servoANGLES[ServoNum_Z];
  //adds each angle depending on the servo number. As
    shown above, depending on the stored variable, each
    angle is added to the empty string.
  //As we stated above, that the array servoANGLES[0-4]
    have their own value for each number, whatever value
    that is gets added to the data. It would look like
    this once all the angles are added to the data:
  //data = "904560120180"
  //to seperate each angle and also to make sure its in
    the right format so the arduino code can translate it
    , we need to add commas after each angle. Which will
    be executed in the next line.
  if (ServoNum_Z < numServos - 1)
   data += ",";
  //after every time a new angle is added, a comma is
    added to the end of each angle.
  //for example: If the ServoNum_Z is equal to zero, that
    means that my angle is 90. It gets added to the
    string "data" and then sees the if statement. the
    numServos is 5, and the last servo is the number 4.
    So i minus the number of servos by 1 to get 4 as my
    value.
  //In this instance, the if statement is met, as 0 < 4 is
     true. it then adds a comma to the string.
  //For example, if the servoANGLES array is at 0, the
    angle is 90. It then gets added to the string so the
    data value would look like this" "90", and then goes
```

```
      through the if condition. If it is true, then it adds a comma so then the updated data value is "90,"
    //It does this for each and every angle until the last one, the maximum value for the final servo is 4. as 4 is not greater than 4, so Processing just sees this as false. It then doesnt add another comma, so the final data will be displayed as this: "90,45,60,120,180"
  }
  dataPort.write(data + "\n");
  //This is the line to end the data to the arduino board, through our serial port that we have stated as dataPort.
  //The data that will be sent is our "data" string that has all our 5 servo angles, the "\n" is for when transmitting data to arduino, the \n tells it that it is the end of the message.

}
```

Arduino Code:

```
    //THIS CODE IS FOR THE SERVO DRIVER MODULE TO READ THE VALUES THAT ARE GIVEN THROUGH THE ARDUINO BOARD

#include <Wire.h>
//allows communication with I2C devices (in this case the servo driver module), the servo driver uses I2C to communicate with the board and code
#include <Adafruit_PWMServoDriver.h>
//adds the library for the operations and commands when using the PCA9685(servo driver)

Adafruit_PWMServoDriver driver = Adafruit_PWMServoDriver();
//a variable that stores the servo driver - "driver"


//Pulses and ticks is what makes the servo driver read on how much the servos should turn,
// For Example:
// pulse of 150 ticks is 0 degrees on the servo
// pulse of 375 ticks is 90 degrees on the servo
// pulse of 600 ticks is 180 degrees on the servo

#define SERVOMIN  150
```

```
//This is defining the MININUM value on how much the servo can turn, so a pulse of 150 ticks is 0 degrees
#define SERVOMAX  600
//This is defining the MAXIMUM value on how much the servo can turn, so a pulse of 600 ticks is 180 degrees

//This part of the code is important, as this is bascially giving the restrictions so the servo driver will read any values between a pulse of 0-600 in ticks. 0-600 / 0-180 degrees

//The servo driver only uses values between 0-600, anything more than 600 will mean that the servo will exceed 180 degrees. Servo motors are only designed for 180 degrees.

int angles[5] = {90, 90, 90, 90, 90};

//This is the inital angle of the servos, this sets all the servos (In this case 5 servos) to 90 degrees at start.

void setup() {
  //this is the setup of the servos and servo drivers before it starts taking data and transmitting it
  Serial.begin(9600);
  // serial refers to the port that is in the arduino, so any data that is given from my laptop will be transferred into the wire and in the Arduino board
  //Serial.begin is setting the communication speed to 9600 baud (bits per second). This baud speed is needed so the arduino board can communicate with any data that my laptop will be giving
  driver.begin();
  //This starts the servo driver to be ready in use, as this begins the I2C communication that was talked about at the beginning of the code
  driver.setPWMFreq(60);
  //hertz is a frequency in which how fast or often the servos recieve control signals.
  //this sets the frequency to 60 hertz, as this is the standard for servos.
  //If I increase the frequency, the servo will get confused and will not operate to its best.
  //If I decrease the frequency, the servo will take longer to respond and also may become slow and jittery. As the communication speed is lacking and isnt efficient
  delay(10);
  //a small delay to give time for the setup to initialize
}

void loop() {
```

```
48 //This is the main code to make the servo turn
49  if (Serial.available()) {
50  //If statement that basically is saying "If there is
       serial data availabe do this", if there isnt then the
       servo wont turn.
51    String data = Serial.readStringUntil('\n');
52    //Reads data from the laptop (Each servo angle value).
         the \n is just indicating to end the reading and to
         store the 5 values for each servo into a string
         called "data"
53    data.trim();
54    //a command that basically removes any extra readings
         that the computer gave the arduino. This is basically
          isolating everything so that we are left with only
         the 5 servo angles, (This removes the "/n" so that
         only the number angles with the commas are left. Ex:
         90,120,30,60,180)
55
56
57    //Main function for this code is to
58    int servo = 0;
59    //sets which servo is being used (5 servos) - servo = 0
         is the 1st servo, servo = 1 is the 2nd servo, servo =
          2 is the 3rd servo, servo = 3 is the 4th servo, and
         servo = 4 is the 5th servo.
60    while (data.length() > 0 && servo < 6) {
61    //data.length repersents the amount of characters in the
          data. For example if I put in 90 degrees for one
         servo, 45 degrees for the 2nd servo, 30 degrees for
         the 3rd servo, 120 degrees for 4th servo, and 180 for
          the 5th servo the data will be stored like this:
         data = (90,45,30,120,180), The total number of
         characters in the data wll be 16.
62    // data.length() > 0 is just saying "While the length of
          characters in the string: "data" is more than 0,
         keep running the code until it is not.
63    //note: && in C++ is a operation which basically means
         that if one side of the conditional statement is
         false, then the whole thing is false. I put this so
         if the index reaches 5 it will not run, and also if
         the length of characters in the string data is 0 it
         will stop working.
64      int commaIndex = data.indexOf(',');
65      //finds the first comma in the data, from my
           processing code, I formatted the servo angles to be
           : angle, angle, angle, angle, angle.
66      //.indexof helps to locate the first comma in the data
           , for example: if my data for the 5 servos are (90,
            45, 120, 180, 60) and these values are stored in
           the string "data" on line 51,the indexof gives me
           the value of where the character is. You can adjust
            what character you are trying to find in the
           brackets, in this case im locating the comma that
           is in the data. Using my example, the comma is the
           2nd character, the number of characters are
           numbered starting from zero.
67      //So, in the example: (90, 45, 120, 180, 60), the data
           .indexOf(',') finds the comma and sets the
           character number into the variable "commaIndex",
           the comma is the 2nd character in the data, as 9 is
            zero 0 is one and then the comma is 2
68      String valueStr;
69      //creates a string variable: "valueStr"
70
71      if (commaIndex != -1) {
72      //If statement that means that if the commaIndex value
            is not equal to -1, run the code that is
           underneath.
73      //i put -1 because the commaIndex will always be above
            0, unless some certain condions are met.
74
75        valueStr = data.substring(0, commaIndex);
76        //.substring() is a function that lets me extract
            part of a string from the entire thing, you put
            your desired start and stop parts in the brackets
             to snip and take out the desired part of the
            string you want. In this case, I have used the
            function for the data value which has all of the
            angles for the 5 servos, this line of code helps
            to actually seperate each angle into its own
            variable. In the brackets I have stated my start
            and stop, 0 to the commaIndex value.
77        //This takes the characters that are from 0 to the
            commaIndex in the string. For example: if our
            comma index was 2 stated above, and our data is
            still (90, 45, 120, 180, 60). It would extract
            all the characters from 0 to 2(the comma). So it
            would extract 90 from the string. The extracted
            value will be then put into a variable called
            valueStr.
78      } else {
79        //This else statement is targetting that if the
            comma index is -1, that means that there are no
            other commas in the data. This part of the code
            was written for when the data is just left with 1
             angle. On line 64 that line of code is for
            locating the comma and setting it as commaIndex.
            When using the .indexOf() function, if there isnt
             a value to be found, it will just return -1.
80        valueStr = data;
```

```
      //The extracted value will be then put into a variable called valueStr.
    }

    angles[servo] = valueStr.toInt();
    //creates an array that converts each servo angle into a integer
    //An array is used to bsacially have multiple variables under one container or bucket, in this case all the variables are integers and each one is seperated into its own corresponding angle value.
    //this array is our final angle value which will be used to translate the integer value into a pulse for the servo driver to understand


    //this part of the code is VERY important, probably the most important piece for the whole entire thing to work. This is what removes the already read angles and proceeds to the next reading for the servo angle.
    if (commaIndex != -1) {
    //if the commaIndex is not equal to -1, meaning, if the comma index is above 0 then do this..
      data = data.substring(commaIndex + 1);
      //Because after every comma, there will be a new angle value, I put plus 1 to change the comma index value to the next number.
      //For example, if we use the data (90,45,120,180,60), we know that our first value for the comma index is 2. Once the program reaches to this area, it will add one to the comma index to start taking the data from the next angle. It is extracted through the .substring() function, as then the new data will be set in the data variable and it would look like this: (45,120,180,60)
      //When using a substring, you do not need to explicitly state your range. If you put in just one value then it will take characters from the value that you set to the end of the string.
    } else {
      data = "";
      //This else statement is for when there are no more commas to be read, that means that there is just one angle left in the string. I put it as an empty string so that in the while loop, the condition of data.length will not be met. This will end the while loop and cause it to not run again.
```

```
     }

     servo++;
     //Adds +1 to the servo variable that was stated right above the while loop, this new servo value is then taken down the same process again.
    }


    for (int ServoNumber = 0; ServoNumber < 6; ServoNumber++) {
    //A for loop is used to keep running a loop a certain number of times, in this case im making it run 5 times. As this part of the code makes the angle values translate to the servo driver module. The servo number starts at 0, indicating that it is talking about 1 out of 5 servos, and then the limit to how many times I want this code to repeat, then lastly to add +1 to the servo number to move on to the next servo
     int angle = map(angles[ServoNumber], 0, 180, SERVOMIN, SERVOMAX);
     //a variable named "angle" that will be our final angle value for the servo, the map() function converts an integer value to the proper pulse in which the servo driver can translate.
     //The format for the map() function is to include your lowest and highest "from" values and also your lowest to highest "to" values. In this map() function, it takes the input angle that was given, (0  180  ) and scales it to the corresponding pulse (150  600   or SERVOMIN and SERVOMAX), It is bascially a ratio.


     driver.setPWM(ServoNumber, 0, angle);
     //This code is for driving each servo, the "driver" was the variable that kept the servo driver module for use. .setPWM() sets the desired pulse, the format for the function in the brackets is: (Pin#, start value, end value)
     //the ServoNumber is for each servo, and the 0 is for keeping the pulse at nothing, the less pulse you have, the less the servo turns. Then the angle is our desired angle that we want which has now been put into a pulse for the servo driver module to read.
     //Then the loop repeats, the ServoNumber increases to 1 and it does the whole thing again. Until, it reaches to 6.
    }
  }
}
```

7. Conclusion:

From the challenging issues with figuring out the power supply of 5 servo motors, to the satisfaction of the sight of seeing my code work. This entire project has taught me so much about computer science, I have attained new information and learned so much. The biggest problem with this project was my inability to get the actual arm assembled. It would have felt satisfying to see that all the hours I put into this project to finally pay off. I had wanted to give myself a real challenge for this final project, to show my dedication and passion for computer science, Arduino, and coding. In result, it took longer than expected for some areas, such as the code itself and also circuitry. When I look back at the beginning of this project, I would say that it has been a success in it teaching me new things that I was not sure I was capable of. It had given me frustration and stress, but also passion and love. In the future, if I get another project like this, I wouldn't challenge myself too much. As even with many hours of me working on this at home, the project still is missing a very important component, Which is the actual robotic arm. Apart from that, I found joy and enthusiasm while working on this project. I would like to thank Mr. Pritchard for making this CS-15 course memorable.

8. Appendix:

1. import processing.serial.* :

   Imports the Serial library to allow Processing to talk to external devices like Arduino.

2. Serial :

   A class that creates a serial communication channel, Ex. USB

3. map() :

   A function that remaps a number from one range to another. Example: angle to slider position.

4. constrain() :

   Keeps a number within a specified range, used for limiting values like servo angles between 0 and 180.

5. mouseDragged() :

   A built-in event that runs while the mouse is being dragged. Used for sliders.

6. include (Wire.h) :

   Allows I2C communication, which is how the servo driver talks to Arduino.

7. include Adafruit PWMServoDriver.h :

   A special library for controlling multiple servos using a PWM driver.

8. driver.begin() :

   Starts I2C communication between Arduino and servo driver.

9. driver.setPWMFreq(60) :

   Sets the signal frequency for servo pulses (60 Hz is standard).

10. .indexOf() :

    a function that finds the position (index) of a specific character or substring inside a string.

11. .substring(start, end) :

    Extracts a piece of a string based on character position.

12. .trim() :

    Removes extra invisible characters like newlines or spaces from incoming data.

13. Servo Driver Module (PCA9685):

    Allows multiple servos to be controlled with just 2 pins from the Arduino. Uses I2C communication.

14. Pulse Width Modulation (PWM) :

    Sends signals to servos based on pulse lengths. 150 = 0°, 600 = 180°.

15. Baud Rate (9600) :

    Speed at which data is sent between Arduino and Processing (9600 bits/sec).

16. I2C Communication :

    A system that lets multiple devices talk to each other over 2 wires. Used for the servo driver.