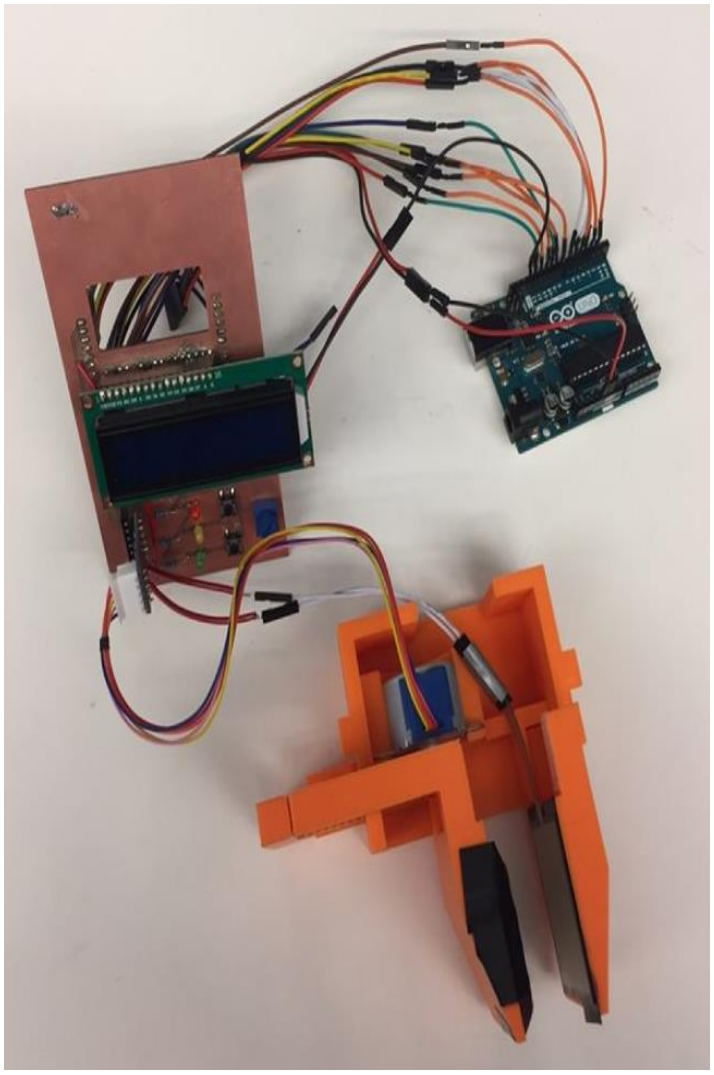
The Claw Arm

Matthew Stanley, Christian Crowley, Yunsik Jung



Introduction

Despite our advancements in robotic and prosthetic technology, there are still many advancements to improve the user's control of the limb. The aim of our project was to create a grasper that could be easily controlled by any individual. The advancement we designed for our gripper is the ability of the arm to detect when it grabbed something. The grasper designed consists of a claw-like appendage actuated by a stepper motor and controlled by two buttons.

Requirements

Since the onset of the project, some requirements had to be met for the project to be considered successful. The claw arm must possess and use at least one sensor and at least one actuator. Further, the circuitry of the grasper must incorporate a custom PCB designed by the project group. Both a pressure sensor and buttons were used to satisfy the sensor requirement of this project. A stepper motor was used as the actuating component for the grasper. Finally, the PCB board was designed in EAGLE and printed on campus. The details of these design choices and their implementation can be found in the following section of this report. The system requirements for the claw arm is that it must be able to grab on object and release it when the user specifies. Since the operator will not be able to feel the object, we would instead display the length of the object we grabbed.

Design and Implementation

The claw arm consists of a housing, a stationary claw segment, a mobile claw segment with an attached rack and pinion drive, the PCB, and an Arduino Uno which operates as the microcontroller of the grasper. Figure 1 below shows a digital rendering of the claw arm and the stepper motor within. All non-electrical components were designed in Solidworks and printed using the 3D printers on campus. The PCB was milled on campus with the aid of Dr. Mcsweeny. All other electrical components were bought by the group members.

Sensors

The pressure sensor is secured to the stationary claw of the claw arm system using a double-sided tape. The choice to use tape was recommended by the manufacture to reduce any stress build up that could occur from glue. The purpose pressure sensor is to determine if something is pressing against it, so these stresses from the glue drying would lead to false readings. The pressure sensor is used to provide a rudimentary tactile sense to the grasper. With this sensor, the claw can not only sense if an object is within its grasp, it also gains some information to the force which it is exerting on the object. The claw also possesses two button sensors which are the primary method of controlling the claw arm. The two buttons are integrated onto the PCB and pass current though an LED and resistor to ensure that the signal was sent to the Arduino board.

Actuators

The stepper motor is fixated within the main claw arm housing as seen in figure 1. A 3D printed gear is fixated to the stepper motor shaft. This gear interfaces with the rack attached to the mobile claw component such that the rotary motion of the stepper motor will move the claw. A schematic detailing the tooth design of the gear and the rack it interfaces with can be seen in figure A-1 and A-2 in the appendix. The stepper motor contains 8 separate, but internal gear box and rack and pinion system provide finer resolution for controlling the claw. By tracking the number of steps taken by the motor, the microcontroller can calculate the size of an object within its grip. Further discussion of this calibration as well as a full discussion of the programming can be found in the code section of this report.

PCB

The PCB for the claw arm was originally designed as a shield that fits on atop of the Arduino board. However, due to a misalignment of some of the pins when the circuit board was milled, jumper cables are instead used to connect the various pins on the board to pins on the Arduino. The PCB also has header pins that allow for the stepper motor and an LCD screen to connect to the board while being replicable should either part breaks. The LCD screen is used to display the current action the claw arm is taking as well as showing the size of objects within the claw's grips. The PCB also provides power to the buttons and pressure sensor and then routes their signals to either digital or analog pins respectively. Figure B-1 in the Appendix B shows a schematic of the PCB designed for the claw arm.

Solidworks Components

In order to accomplish the purpose of the project, the claw was designed in Solidworks and 3D printed on campus. The claw arm consists of the main body, two claw components, spur gear, and step motor. A pinion and rack gear set work to convert the rotational motion into the linear motion for driving the claw. The assembly model of the claw arm is shown in Figure 1. When the gear rotates clockwise, the claw system opens; and the system closes when the gear rotates counter-clockwise. When it comes to the design of spur gear and rack gear, we followed to the standard criteria and the M1 module of gear was applied. Further detail on the design of the pinion and rack gear system, as well as the models for each part of the design, can be found in Appendix A.

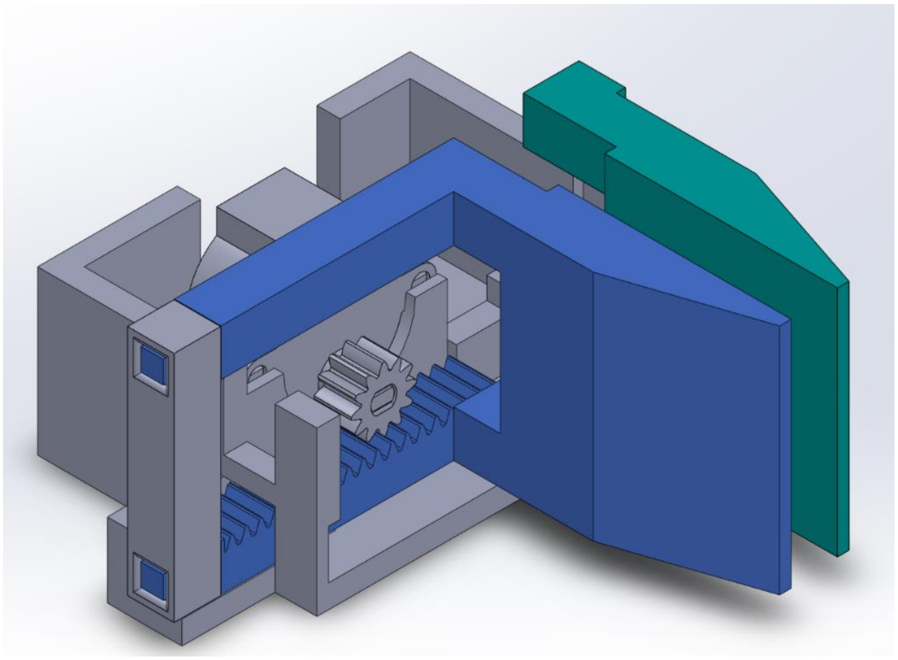


Figure 1. Assembly model of the claw arm

Programming

Our code is broken into 4 sperate states that dictate how the claw operates. These states are calibration, stationary, open, and close. While this sections discusses the functionality of the code, the complete pseudocode can be found in Appendix C and the actual Arduino coding can be found in Appendix D.

When first powered on, the claw arm starts in its calibration sate. In this state, the claw arm determines its valid range of motion. First, the LCD screen informs the user that they should clear the area around the claw. This is to prevent the claw from closing on anything during the calibration and wrongfully setting the zero value. Once the claw has closed fully, the pressure sensor will sense contact between the two claw components and set its current position as its zero-displacement position. Next, the motor will move a predetermined distance to its fully open position. After this the microcontroller switches to the stationary state.

The stationary state has two duties. When first entered, the microcontroller computes the distance between the two claw components based on the number of steps the motor is from the zero-value founded during calibration. The value is then displayed on the LCD screen. The other action this state performs is to listen for a press of the open or close button on the PCB. Pressing the open or closed button will send the craw arm into the open or close state respectively.

The open and close states perform very similar tasks. Their task is to indicate to the user which state they are currently in and to open or close the claw. The stepper motor will continuously tick into the desired direction until certain conditions are met; the first condition being that the claw remains with the boundaries constructed in the calibration step. The second condition is that the user presses the button opposite from the state which it is currently in. The closed state has a third condition that listens for a desired pressure threshold. When any of these three conditions are met, the claw arm returns to the stationary state.

Calibration

For the claw arm to meet the system requirements, three parameters need to be calibrated: the pressure sensor reading, the step range that the claw can travel, and the conversion between the stepper motor's steps and the distance that the claw moved. While we did not need to convert the digital reading into a pressure reading, we needed to determine an acceptable tolerance for when the claw thought it grabbed something. We saw digital noise of about 10, but a light touch would increase the reading by 100. After some trial-and-error, we determine that the threshold reading should be 250. The maximum steps the arm can take from zero was calculated by knowing the motor rotation to rack gear ratio. We found that the claw arm would move one tooth on the rack for about 320 steps that the motor takes. From the zero position, the rack had seven teeth to move with, resulting in a max range of 2240. Not wanting to get too close to the limits, we settled on setting the upper count limit to 2000.

The final item to calibrate was the conversion between the step values and distance. Since the pinion and rack gear system moves proportional to the number of steps taken by the motor, we expected to find a linear relationship. We then started taking measurements of objects we found in the room and displayed the tick value when the arm grabbed the object. The results are shown, along with the best fit line, in figure 2.

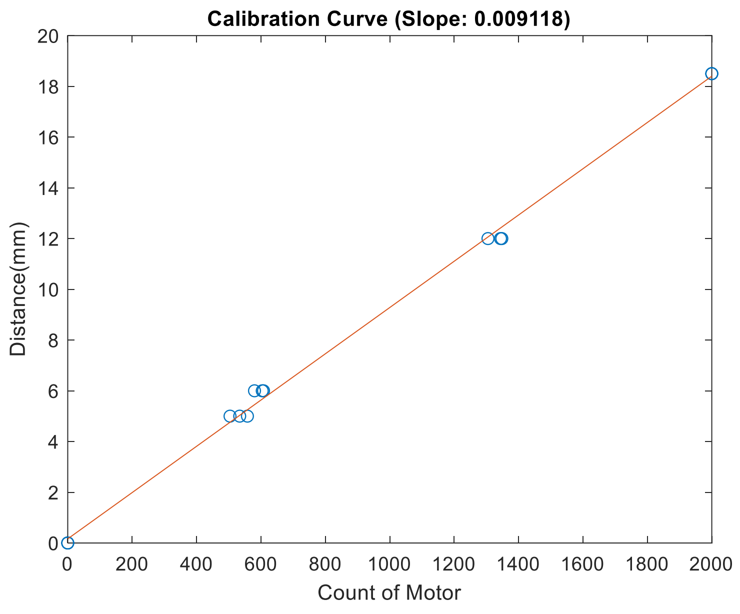


Figure 2. Plot of the relationship between step count and distance in millimeters

Work division breakdown

The work for this project was broken into three design sections focused on mechanical, PCB, or coding aspects of the claw. The mechanical design was headed by Yunsik and encompassed the Solidworks design of the claw and the gears needed for transferring motion from the motor to the claw. The PCB, lead by Christian, section focused on the design of the circuit board in EAGLE as well as milling the board in an on-campus lab. Finally, Matthew was the main coder for the project.

Risks and problem

During the process of making the claw arm, we ran into a few unexpected problems. When it comes to the 3D printing, we needed to find appropriate properties for setting up the printer we used, otherwise the printer produced low quality features. The printer was had lower tolerances than expected, leading to prints of different quality every time. In order to overcome the problem, we printed several different sizes of many of the components and found the one that fit best for the system.

The assembly of the PCB also presented some major challenges for this group. EAGLE assumes that all header pins work as vias when placing leads in the software. This led to issues where the header pins would not connect to the opposite side it was soldered onto. Because of this, many hours were wasted unsoldering the components and raising them so that solder could be added to pads on both sides of the boards.

In our proposal it was mentioned that this claw arm was meant to be arm mounted. While that is still the final goal of the claw arm concept, it was unnecessary to the system requirements that the claw has an arm mount as it is primally a cosmetic feature. One repercussion of this change is that the circuit board and LCD screen are now free-floating as they would have been attached to this arm mount.

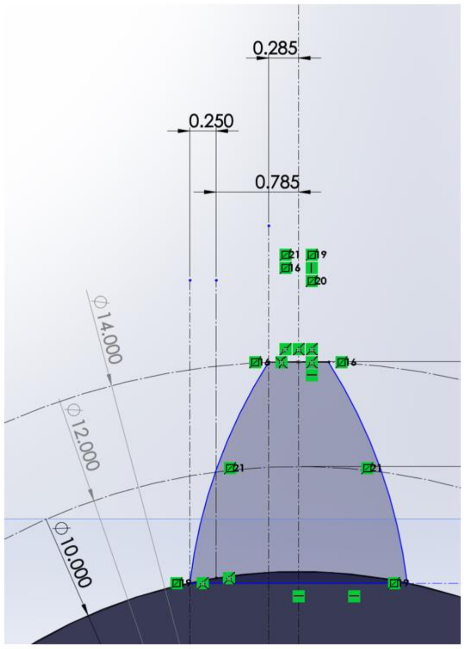
Conclusions

The claw arm designed consists of main body, two claw components, and gear system. A step motor was used for actuation of the system. Buttons were used to control the system while a pressure sensor was utilized to tell if anything was in the grasp of the claw. Finally, an LCD screen displays the size of objects grabbed by the claw as well as the current state that the claw is in.

To meet the project requirement, the claw arm designed has two types of sensors and one external effector. Further, it incorporates a designed PCB and uses a microcontroller with embedded code. Finally, the claw arm works as intended and can perform the tasks laid out in the proposal.

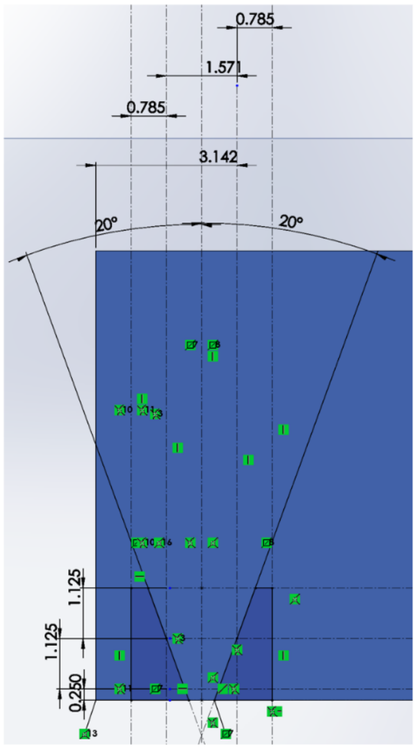
While the claw arm designed is a success, there is still room for future work to further develop this project. An arm mount still needs to be designed so that the claw can easily be affixed to a residual limb. The PCB also needs to be reduced in size as its current dimensions would make it difficult for use on a prosthetic. Finally, the micro-controller should be integrated within the PCB itself, instead of as an external attachment.

Appendix A: Mechanical Drawings



Spur Gear	
Module	1
Number of teeth	12
Pitch Circle Diameter	12 (mm)
External Diameter	14 (mm)

Figure A-1. Spur gear property (unit: mm)



Rack Gear	
Module	1
Height	2.5 (mm)
Height of end teeth	0.25 (mm)
Pressure angle	20°

Figure A-2. Rack gear property (unit: mm)

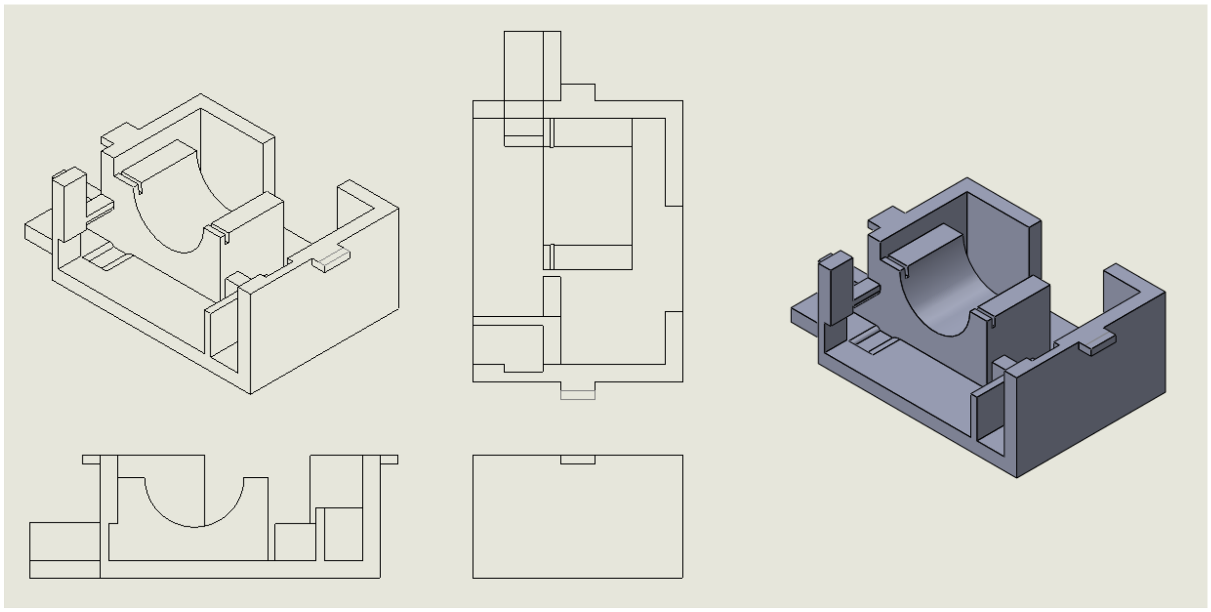


Figure A-3. Drawing and isometric view of main body

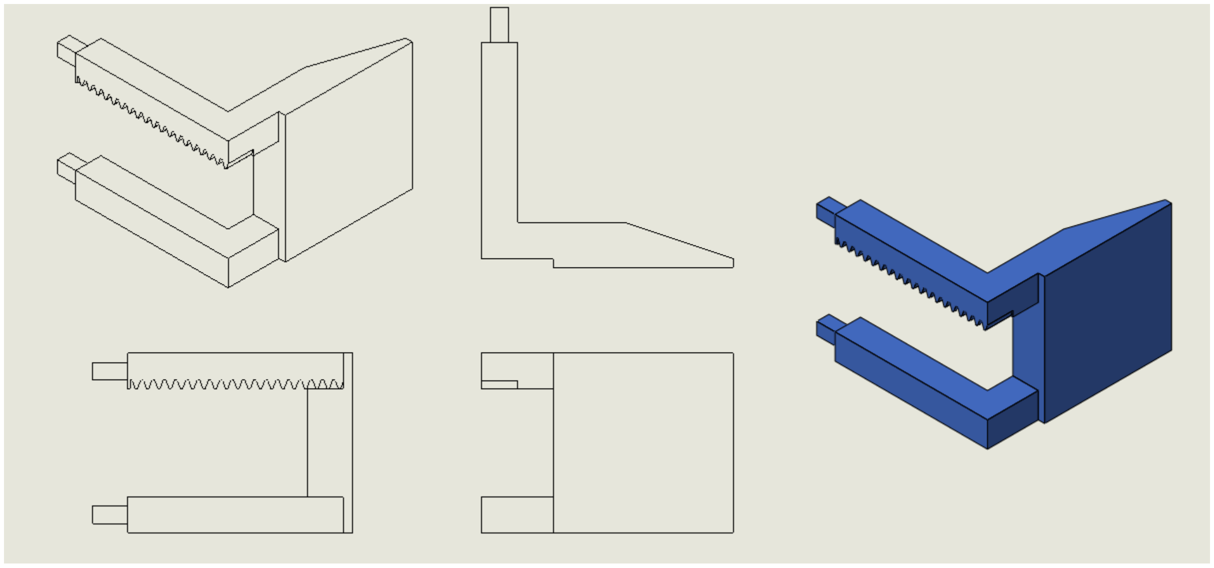


Figure A-4. Drawing and isometric view of left claw

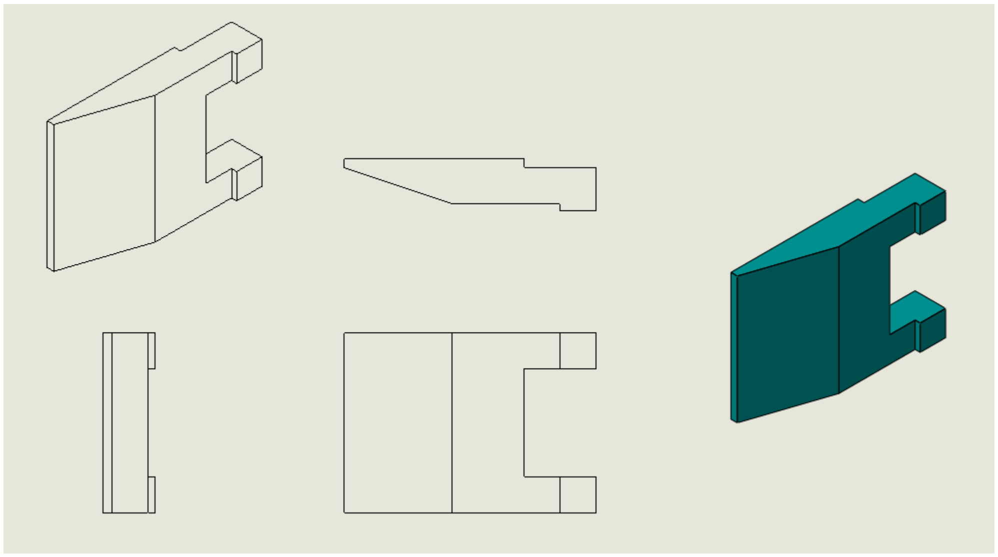


Figure A-5. Drawing and isometric view of right claw

Appendix B: Electrical Schematic

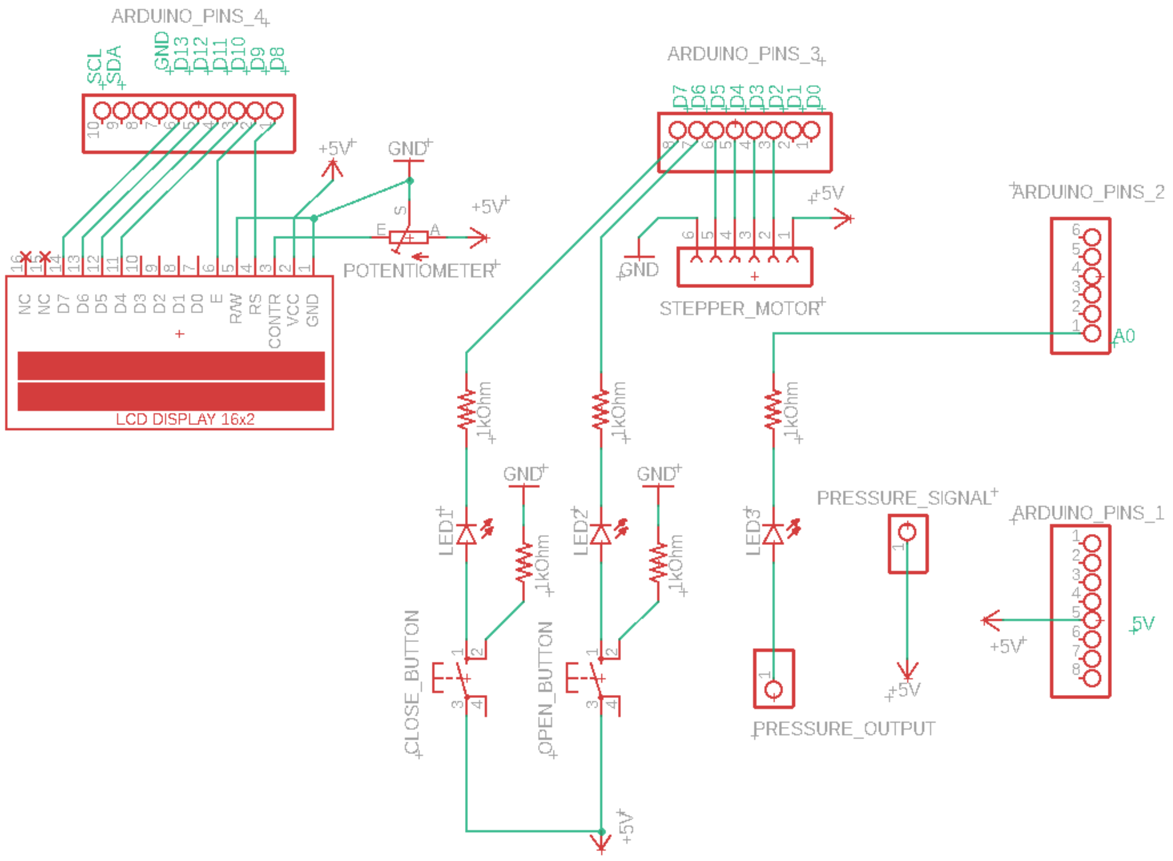


Figure B-1. PCB Schematic

Appendix C: Pseudocode

```
int tick: This variable stores the current
uint8 state: Keeps track of the current state
(0=Calibration,1=Close,2=Open,3=Stationary)
const int maxTicks: This variable stores the max number of steps
motor and move away from 0

main():
Set variable ‘gotZero’ to false and ‘state’ to 0. Run the rest
in a while loop.
  • If ‘state’ is 0, check value of ‘gotZero’
      o If ‘gotZero’ is true, check if ‘tick’ is equal to
        ‘maxTicks’
          ▪ If equal, set ‘state’ to 3
          ▪ Else, run ‘tickCounter(&tick)’
      o Else, run and check the value of ‘isGrabbed()’
          ▪ If ‘isGrabbed()’ is true, set ‘tick’ to 0 and
            ‘gotZero’ to true
          ▪ Else, run ‘tickClockwise(&tick)’
  • If ‘state’ is 1, run and check the value of ‘isGrabbed()’
      o If ‘isGrabbed()’ is true, set ‘state’ to 3
      o Else, run ‘tickClockwise(&tick)’
      o If ‘pressedOpen’ is true, set ‘state’ to 3
  • If ‘state’ is 2, check if ‘tick’ is equal to ‘maxTicks’
      o If equal, set ‘state’ to 3
      o Else, run ‘tickCounter(&tick)’
      o If ‘pressedClose’ is true, set ‘state’ to 3
  • If ‘state’ is 3, calculate the displacement of the claw
    using ‘tick2distance(tick)’.  Display the result.  Then
    check if a button has been pressed.
      o If ‘pressedClose’ is true, set ‘state’ to 1
      o If ‘pressedOpen’ is true, set ‘state’ to 2

tickClockwise(int* tick):
Rotates the stepper motor one step clockwise
INPUT: tick stores the current number of steps from closed
OUTPUT: NA

tickCounter(int* tick):
Rotates the stepper motor one step counter-clockwise
```

INPUT: tick stores the current number of steps from closed
OUTPUT: NA

isGrabbed():
Check if there is enough pressure for grabbing an object by reading the value of the pressure sensor.
INPUT: NA
OUTPUT: Boolean if enough pressure on sensor (1) or not (0)

tick2distance(const int tick):
Converts a step value of the stepper motor into distanced opened by the claw arm through knowing the dynamics of the claw.
INPUT: tick stores the current number of steps from closed
OUTPUT: the current distance the claw arm is from being closed in __units__

pressedOpen():
Checks if the button has been pressed indicating to open the claw.
INPUT: NA
OUPUT: Boolean if the button has been pressed (1) or not (0)

pressedClose():
Checks if the button has been pressed indicating to close the claw.
INPUT: NA
OUPUT: Boolean if the button has been pressed (1) or not (0)

Appendix D: Arduino Code

```
#include <LiquidCrystal.h>

// System Setup
uint8_t state = 0; // 0 Calibration, 1 Close, 2 Open, 3 Stationary
boolean change = true;

void stepper(long tick);
float tick2distance(long tick);

// Display Setup
LiquidCrystal lcd(8,9,10,11,12,13);

// Stepper Setup
#define IN1 5
#define IN2 4
#define IN3 3
#define IN4 2
long maxTick = 2000;
long tick = maxTick;
uint8_t offset = 0;
boolean foundMin = false;

//Pressure Sensor
#define PRESSURE A0
int maxPressure = 250;
int reading;

// Buttons
#define OPEN 6
#define CLOSE 7

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  lcd.begin(16,2);
  lcd.print("The Claw");
  Serial.println("The Claw");

  pinMode(IN1,OUTPUT);
  pinMode(IN2,OUTPUT);
  pinMode(IN3,OUTPUT);
  pinMode(IN4,OUTPUT);

  pinMode(OPEN,INPUT);
  pinMode(CLOSE,INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  switch(state){

    case 0: // Calibration
      if (change){ // Display State
        lcd.setCursor(0,0);
        lcd.print("Calibration     ");
        lcd.setCursor(0,1);
        lcd.print("Clear Workspace ");
        change = false;
      }

      if (!foundMin){ // Close until max is found
        reading = analogRead(PRESSURE);
        //Serial.println(reading);
        if (reading >= maxPressure){
          foundMin = true;
          offset = tick & 7;
          tick = maxTick;
        }else{
          ++tick;
          stepper(tick);
        }
      }else{ // Open until max is found
        if (tick <= 0){
          state = 3;
          change = true;
        }else{
          //Serial.println(tick);
          --tick;
          stepper(tick+offset);
        }
      }
      break;

    case 1: //Close
      if (change){ // Display State
        lcd.setCursor(0,0);
        lcd.print("Closing Claw    ");
        change = false;
        Serial.println("CLOSING");

      }

      // Interrupt and stop closing
      if (digitalRead(OPEN) || (analogRead(PRESSURE) >= maxPressure)
|| tick >= maxTick){
        state = 3;
        change = true;
      }
      else{
        ++tick;
        stepper(tick + offset);
      }
      break;

    case 2:// Open
      if (change){ // Display State
        lcd.setCursor(0,0);
        lcd.print("Opening Claw    ");
        Serial.println("OPENING");
        change = false;
      }

      // Interrupt and stop opening
      if (digitalRead(CLOSE) || (tick <= 0)){
        state = 3;
        change = true;
      }
      else{
        --tick;
        stepper(tick + offset);
      }
      break;

    case 3:
      if (change){ // Display State
        lcd.clear();
        lcd.print("Displacement");
        lcd.setCursor(0,1);
        lcd.print(tick2distance(tick));
        change = false;
      }

      // Check if should open or close
      if (digitalRead(CLOSE)){
        state = 1;
        change = true;
      }
      if (digitalRead(OPEN)){
        if (change){
          state = 3;
          change = false;
        }else{
          state = 2;
          change = true;
        }
      }

      break;

    default: // Something went wrong
      state = 0;
      Serial.print('e');
```

```
        Serial.println('1');
        break;
    }
    Serial.println(analogRead(PRESSURE));
  }

  void stepper(long tick){
    /* Update step value */
    uint8_t seq = tick & 7;
    switch(seq){
      case 0:
        digitalWrite(IN1,HIGH);
        digitalWrite(IN2,LOW);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,LOW);
        break;
      case 1:
        digitalWrite(IN1,HIGH);
        digitalWrite(IN2,HIGH);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,LOW);
        break;
      case 2:
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,HIGH);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,LOW);
        break;
      case 3:
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,HIGH);
        digitalWrite(IN3,HIGH);
        digitalWrite(IN4,LOW);
        break;
      case 4:
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,LOW);
        digitalWrite(IN3,HIGH);
        digitalWrite(IN4,LOW);
        break;
      case 5:
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,LOW);
        digitalWrite(IN3,HIGH);
        digitalWrite(IN4,HIGH);
        break;
      case 6:
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,LOW);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,HIGH);
        break;
      case 7:
        digitalWrite(IN1,HIGH);
        digitalWrite(IN2,LOW);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,HIGH);
        break;
      default:
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,LOW);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,LOW);
        break;
    }
    delay(1);
  }

  float tick2distance(long tick){
    /* Converts a step count to a position */
    return 0.009118*(maxTick - tick);
  }
```