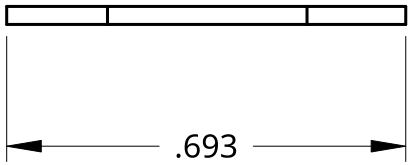
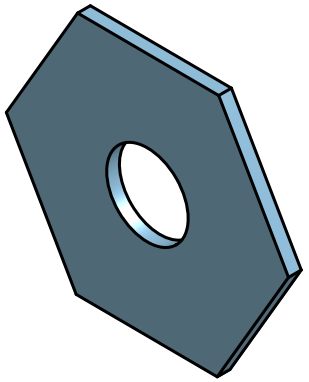
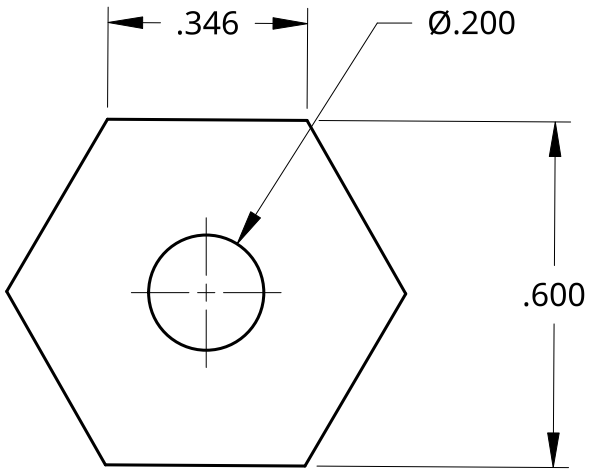
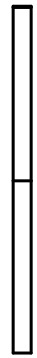


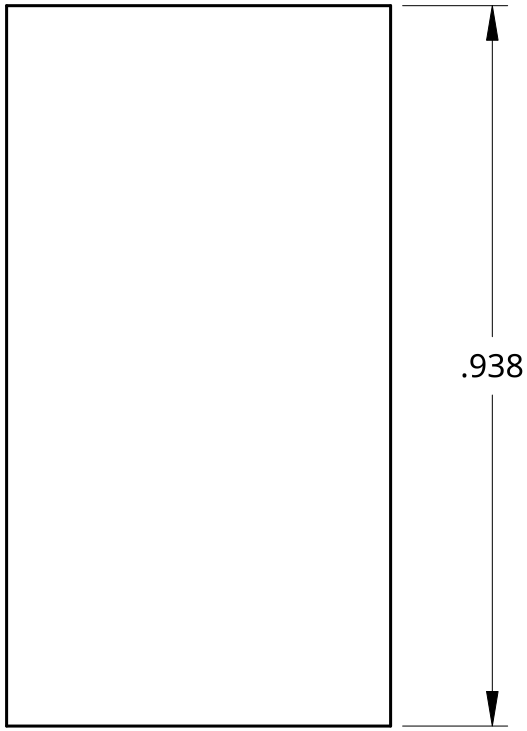


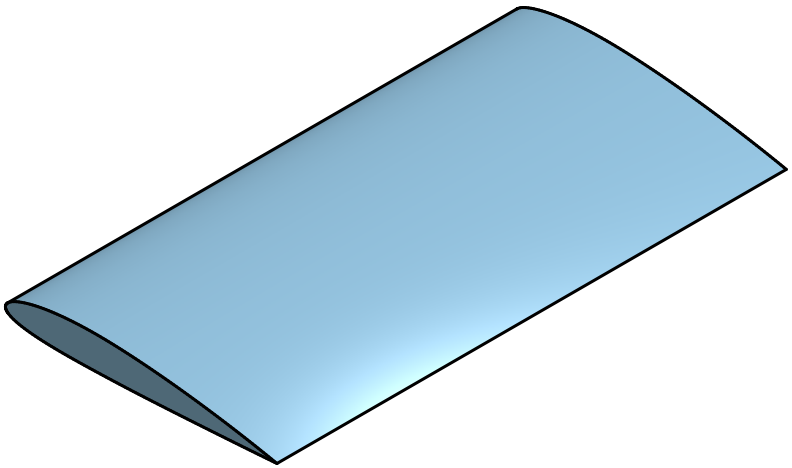


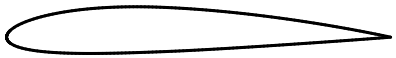


UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	08/16/2024
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

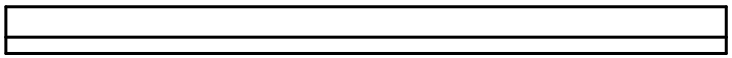
TITLE		
SIZE A	DWG NO.	REV.
SCALE 3:1	WEIGHT	SHEET 1 of 1





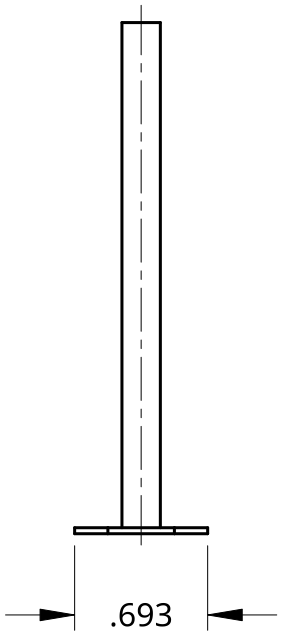


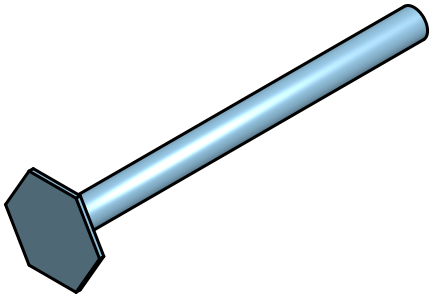
NACA 2412

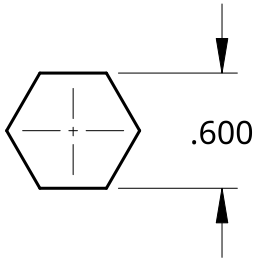


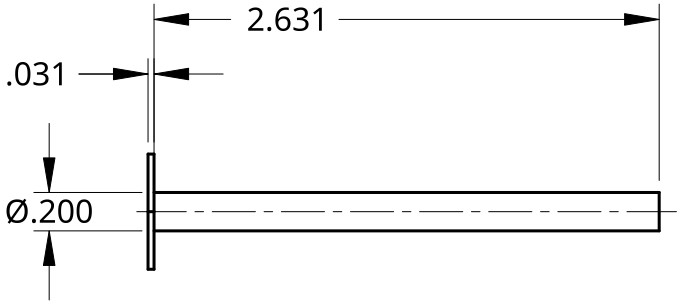
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	08/16/2024
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE		
SIZE A	DWG NO.	REV.
SCALE 4:1	WEIGHT	SHEET 1 of 1



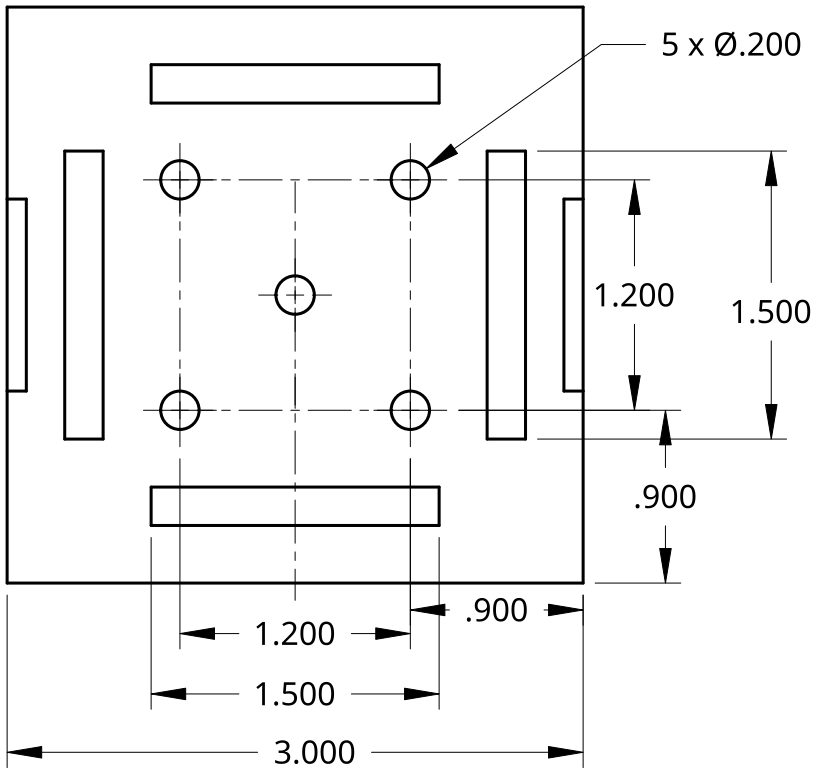


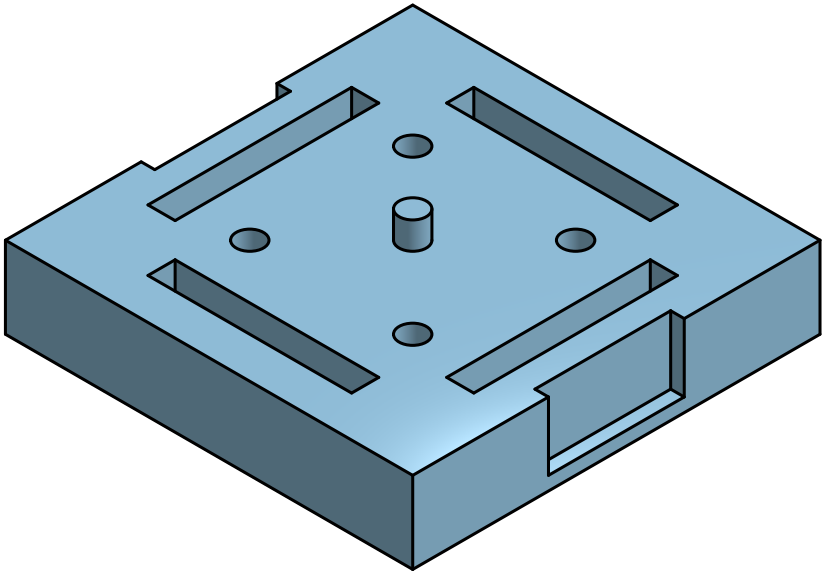


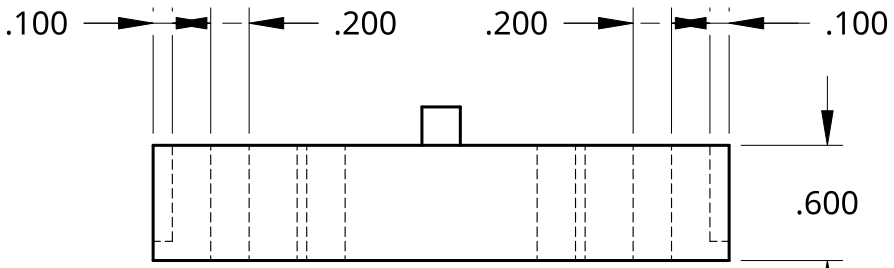


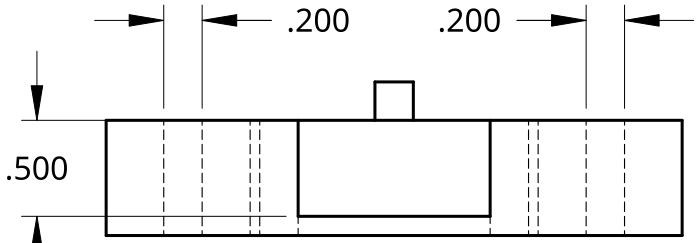
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	08/16/2024
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE		
SIZE A	DWG NO.	REV.
SCALE 1:1	WEIGHT	SHEET 1 of 1



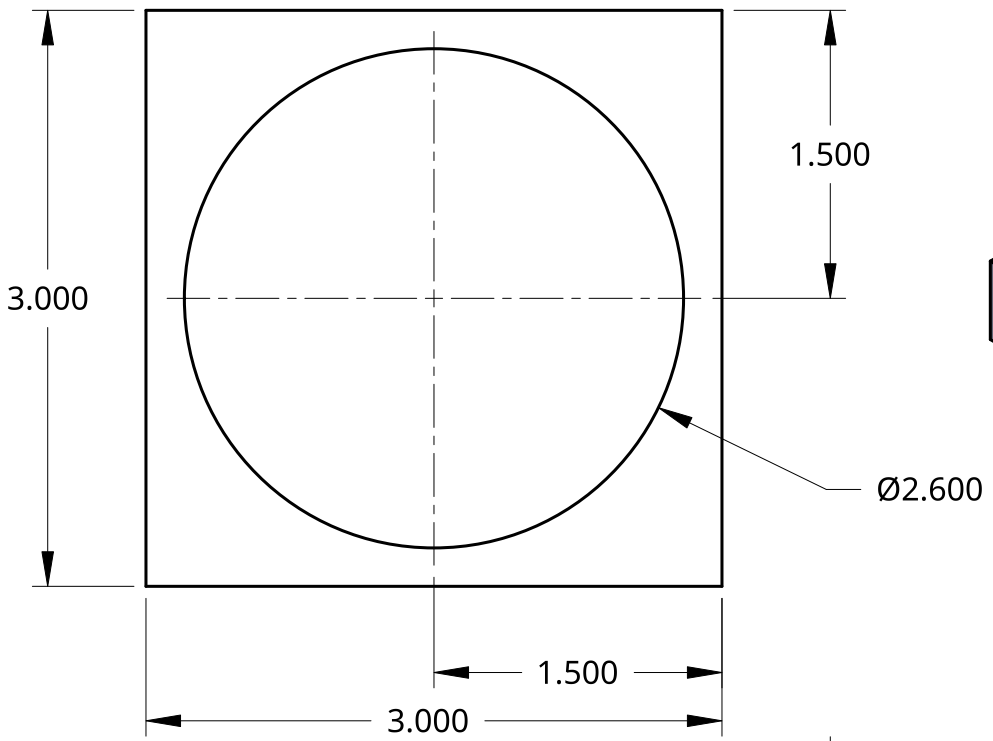


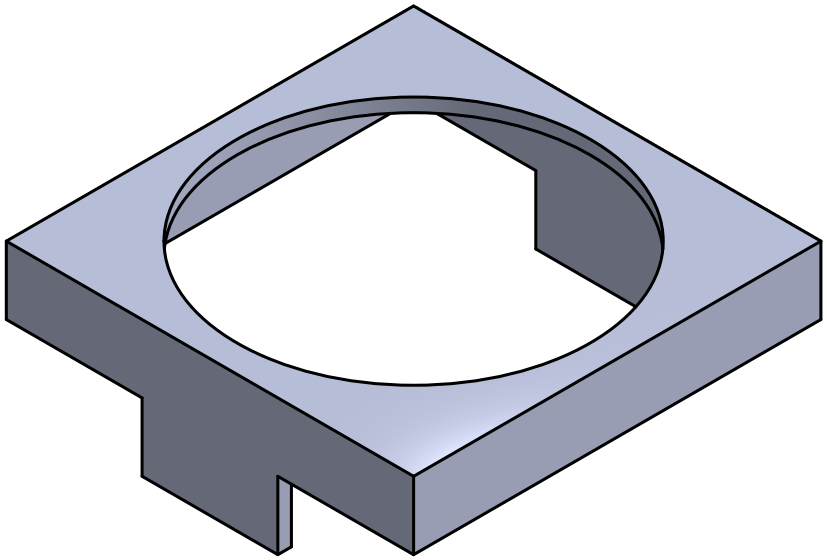


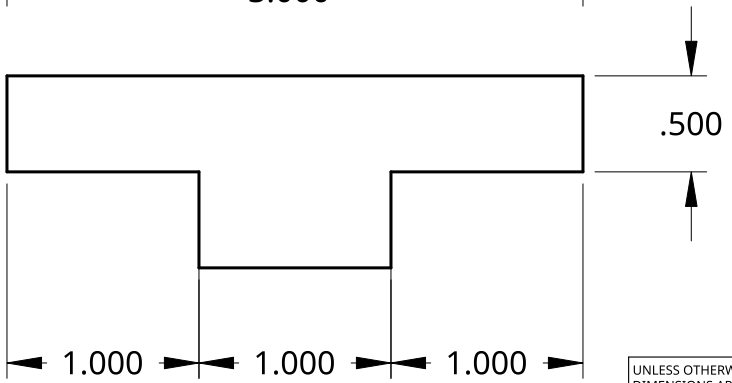


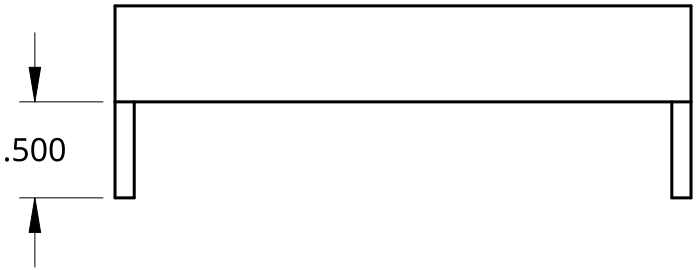
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	08/16/2024
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE		
SIZE A	DWG NO.	REV.
SCALE 1:1	WEIGHT	SHEET 1 of 1



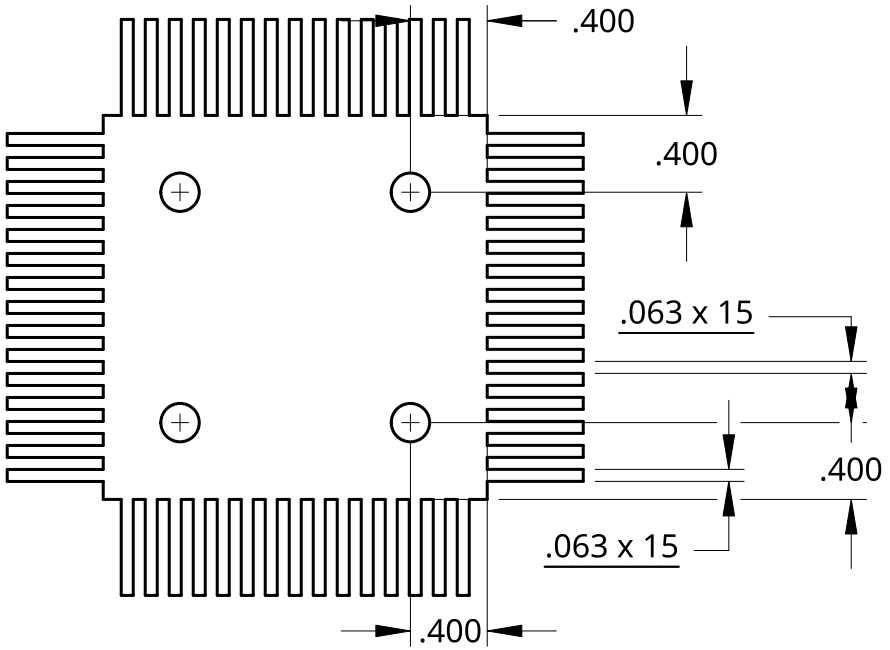


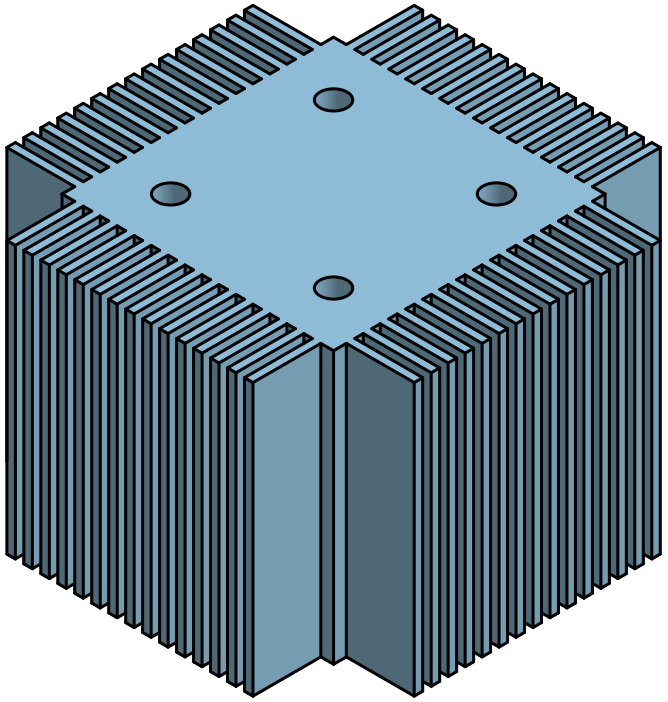


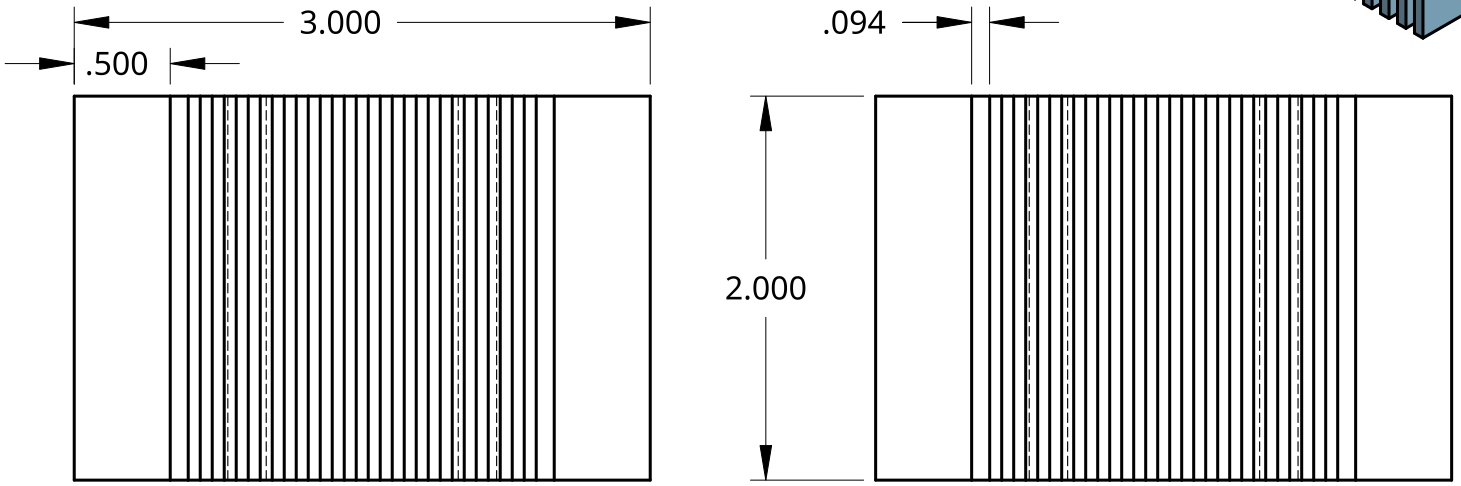


UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	08/16/2024
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE		
SIZE A	DWG NO.	REV.
SCALE 1:1	WEIGHT	SHEET 1 of 1

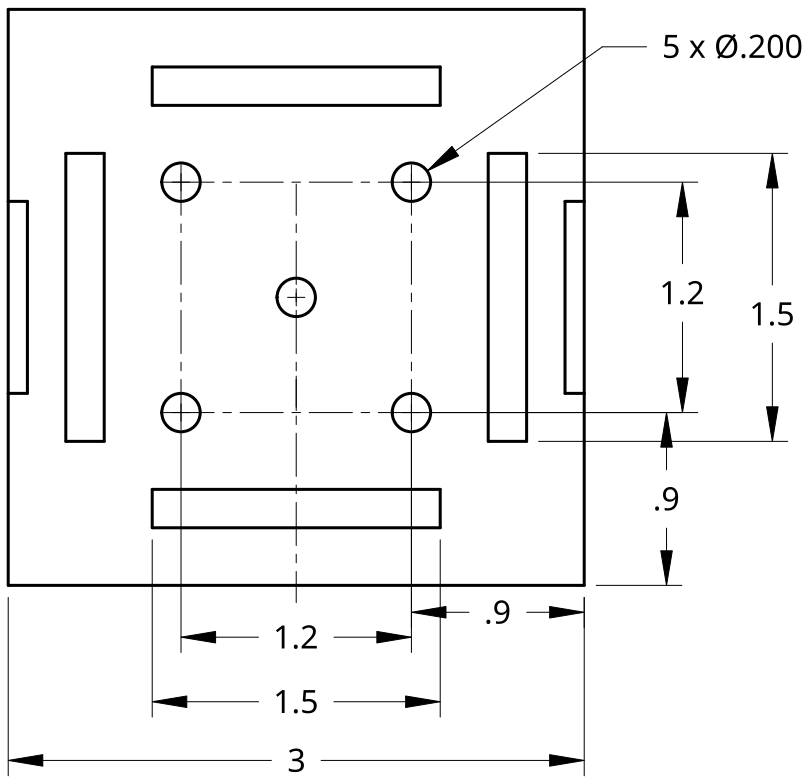


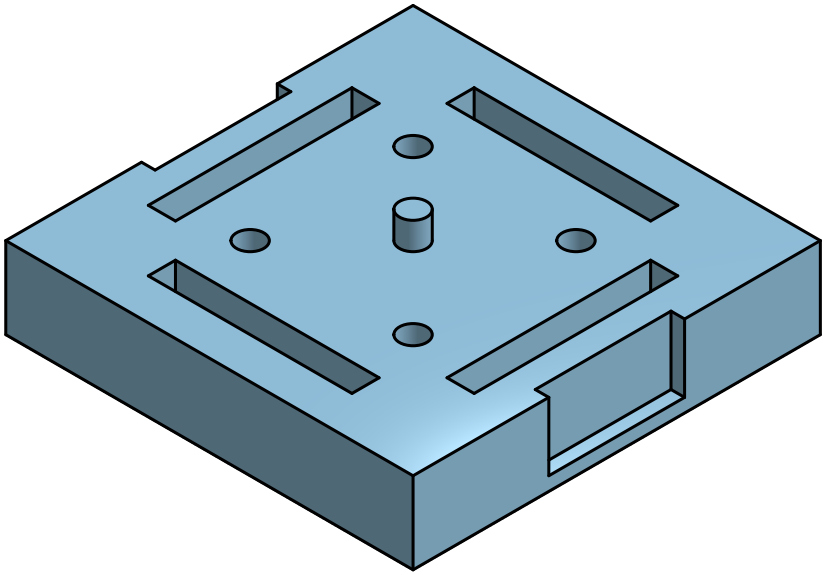


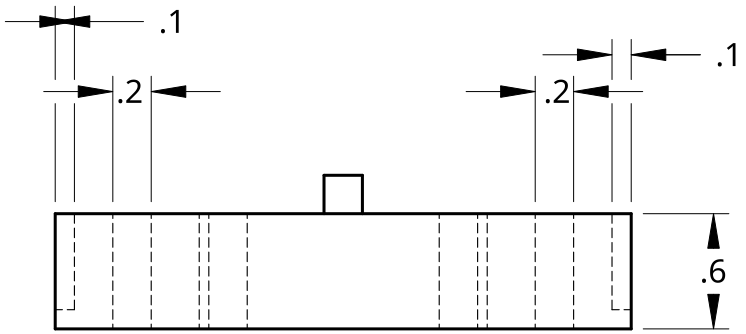


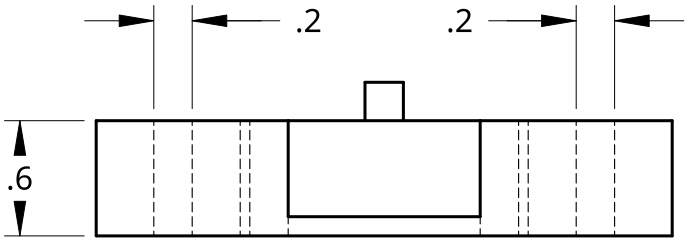
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	05/10/2022
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE		
SIZE A	DWG NO.	REV.
SCALE 1:1	WEIGHT	SHEET 1 of 8

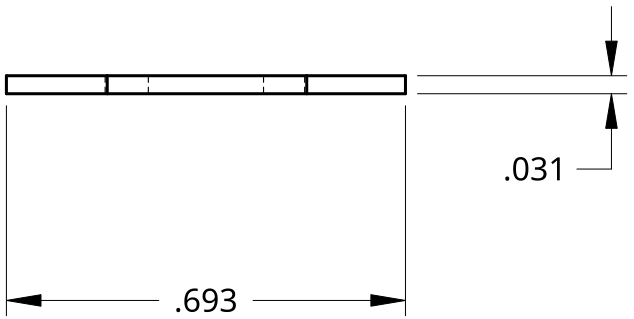


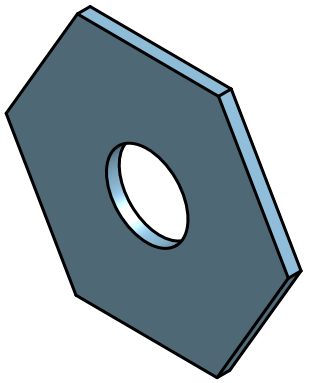


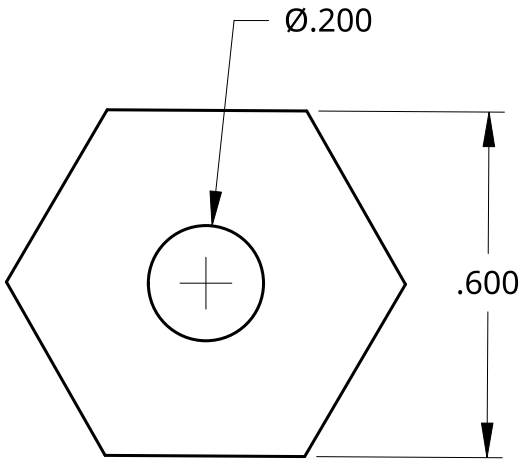




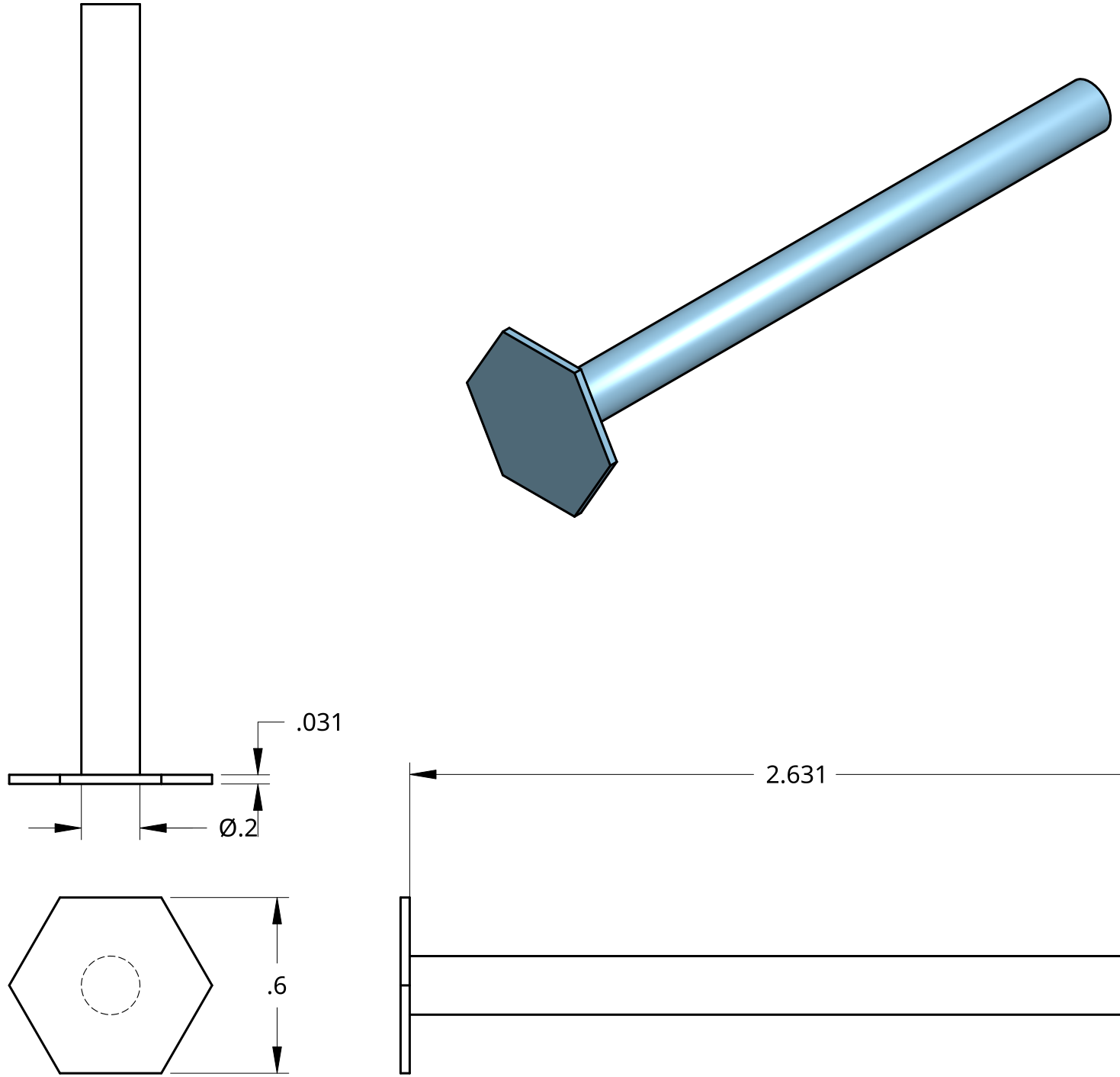
DWG NO.	SIZE	SCALE	SHEET	REV.
	A	1:1	2 of 8	



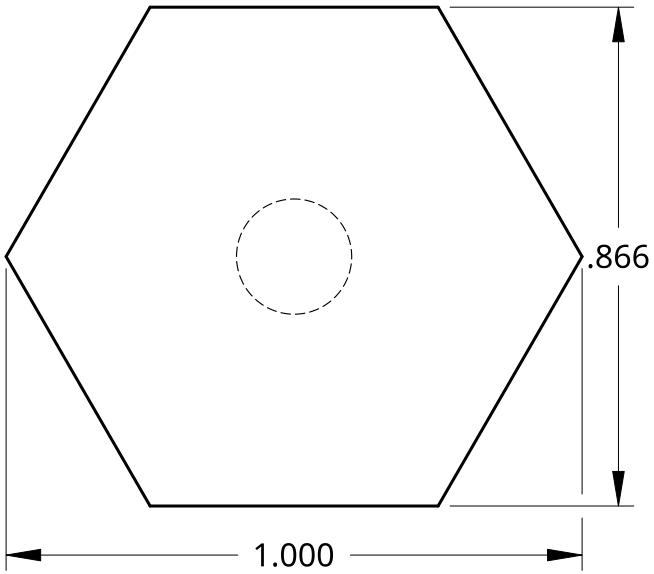


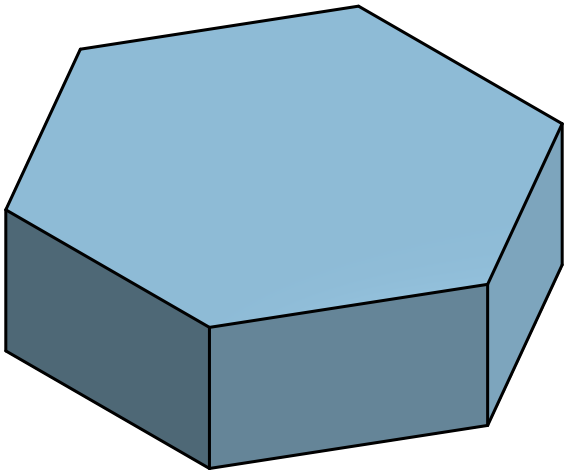


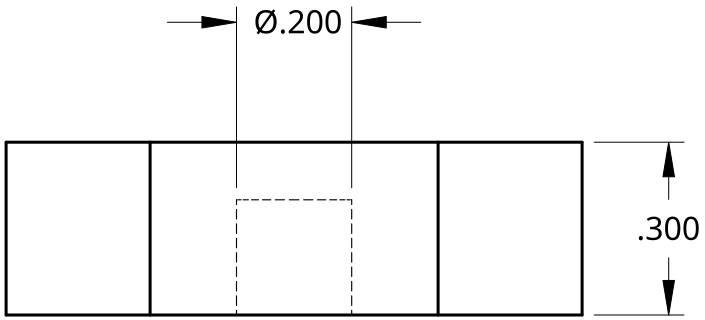
DWG NO.	SIZE	SCALE	SHEET	REV.
	A	3:1	3 of 8	

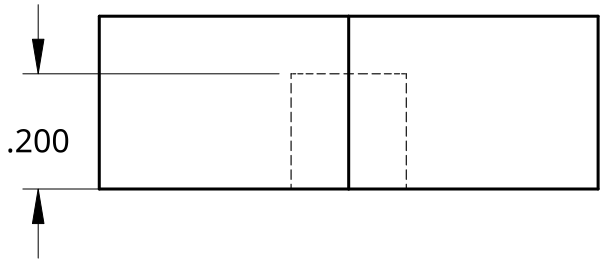


DWG NO.	SIZE	SCALE2	SHEET	REV.
	A	2:1	4 of 8	

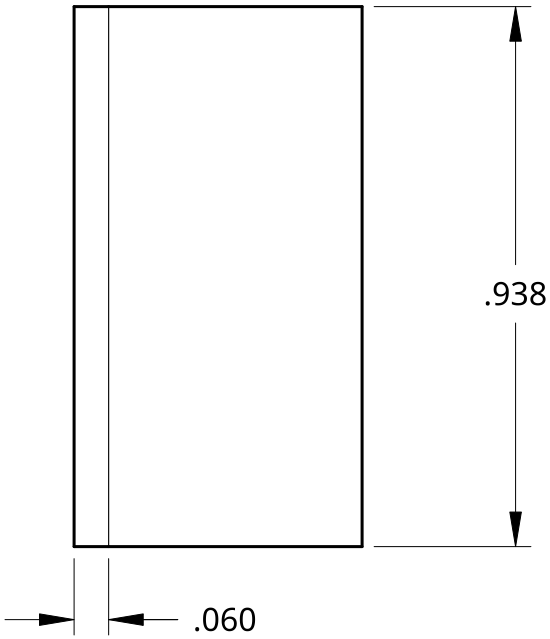


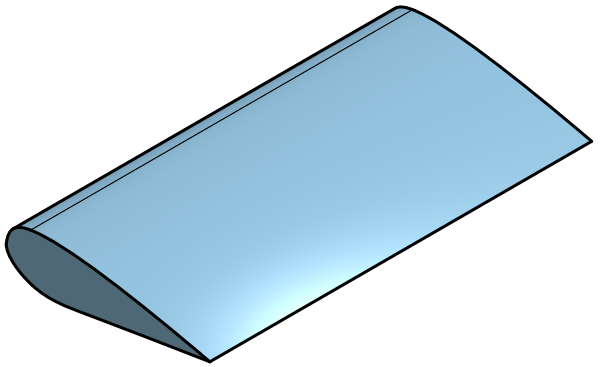




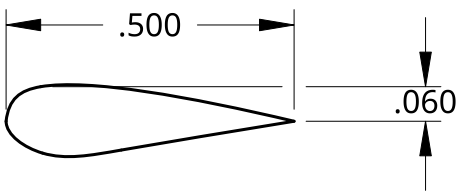


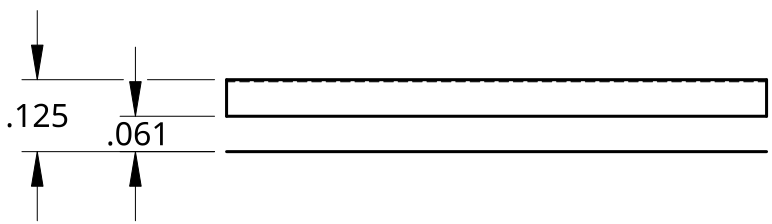
DWG NO.	SIZE	SCALE	SHEET	REV.
	A	3:1	5 of 8	



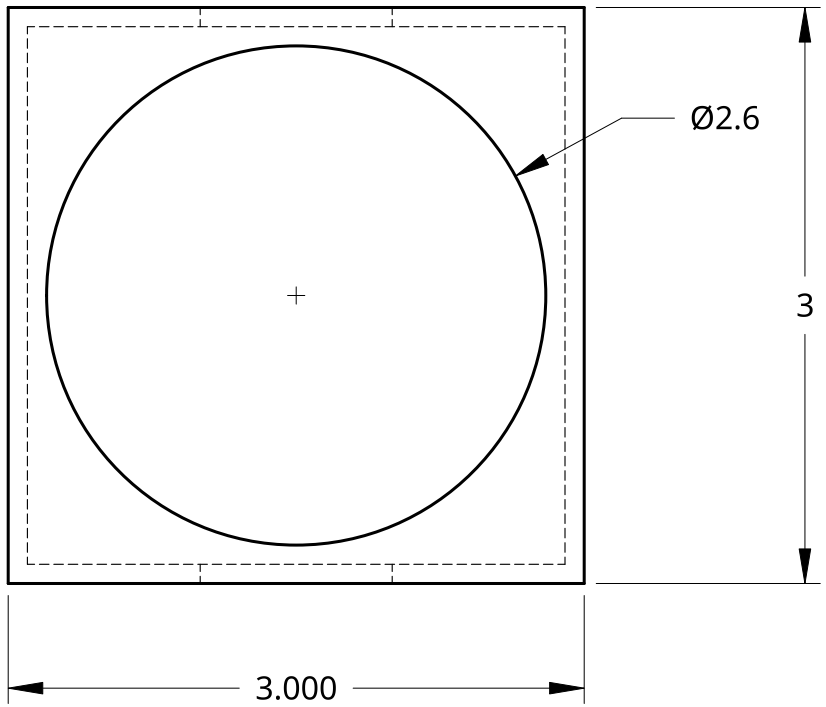


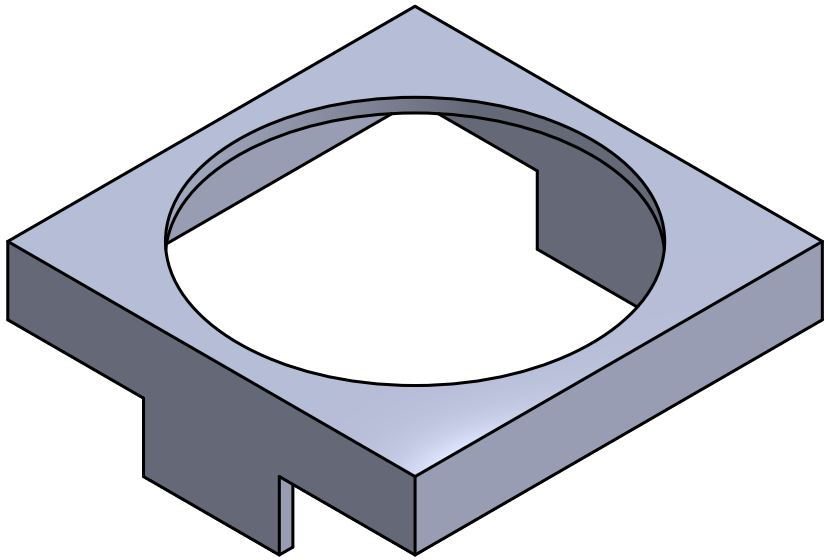
NACA
2412

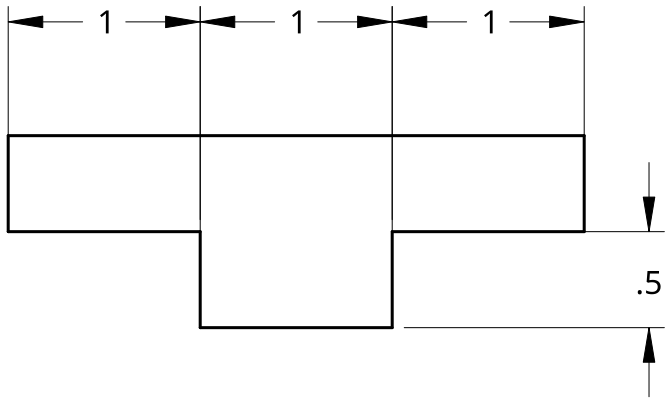


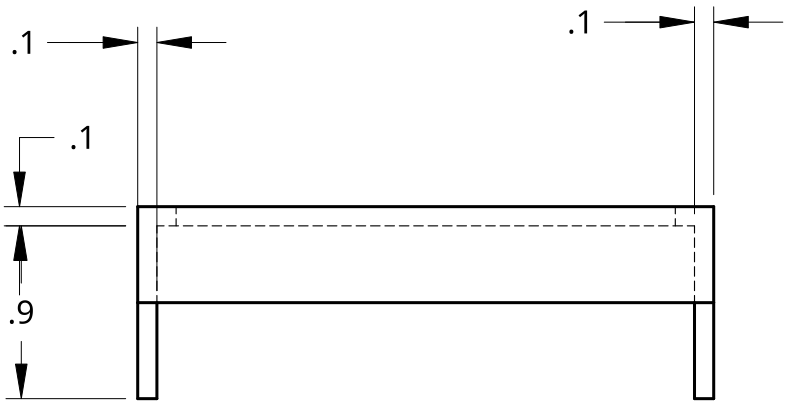


DWG NO.	SIZE	SCALE	SHEET	REV.
	A	3:1	6 of 8	

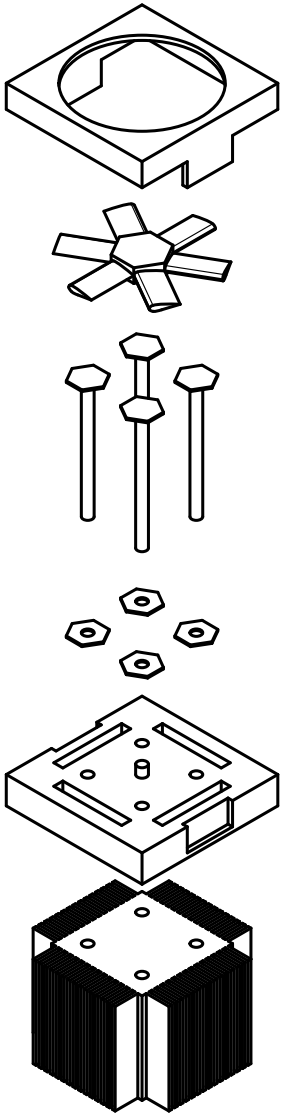


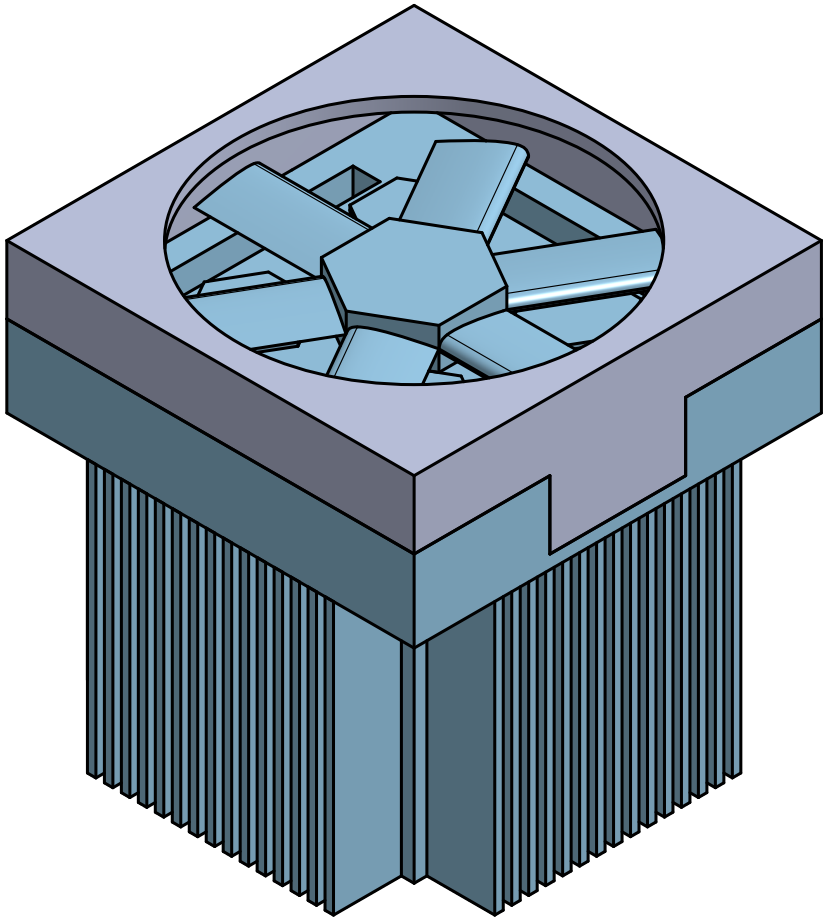






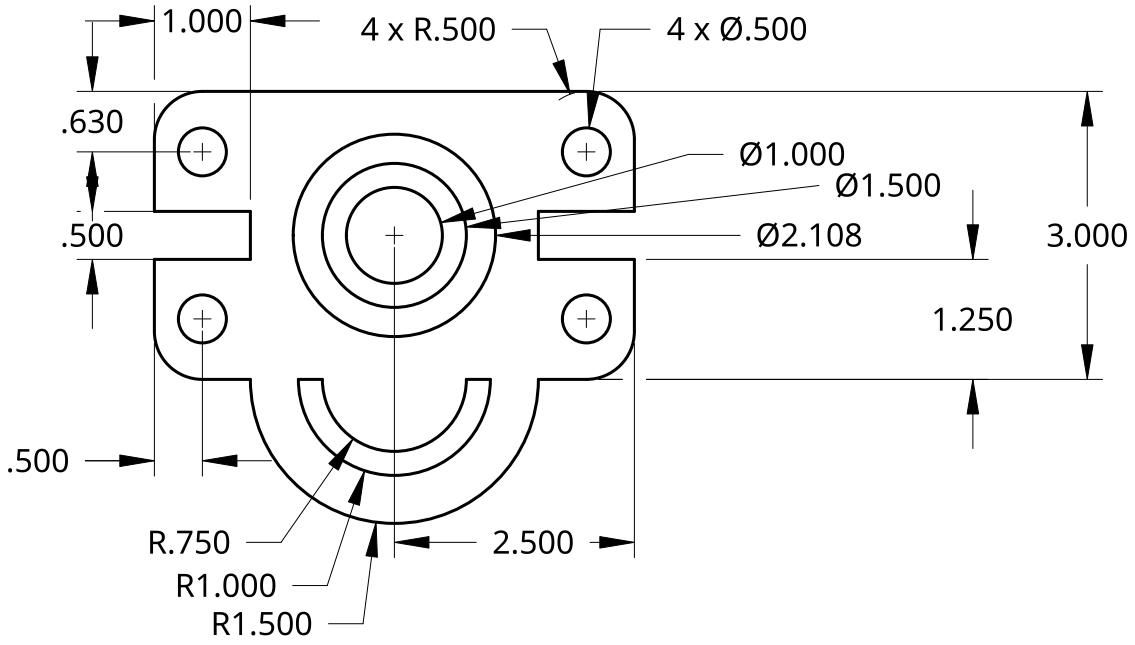
DWG NO.	SIZE	SCALE	SHEET	REV.
	A	1:1	7 of 8	

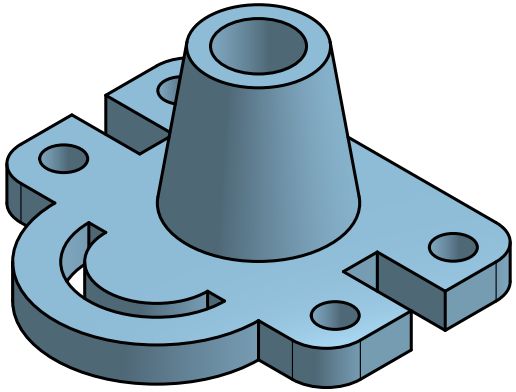


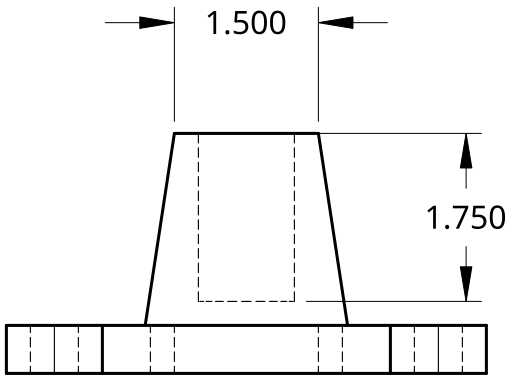


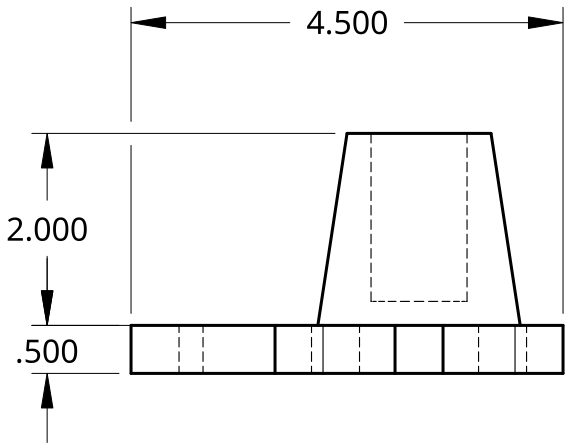
SCALE 1:1

DWG NO.	SIZE	SCALE	SHEET	REV.
	A	1:3	8 of 8	



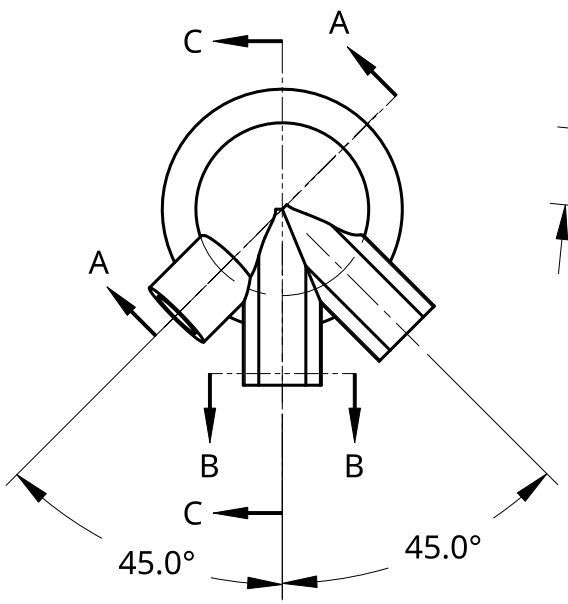


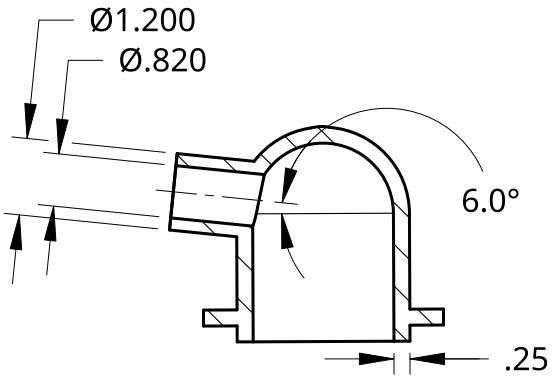




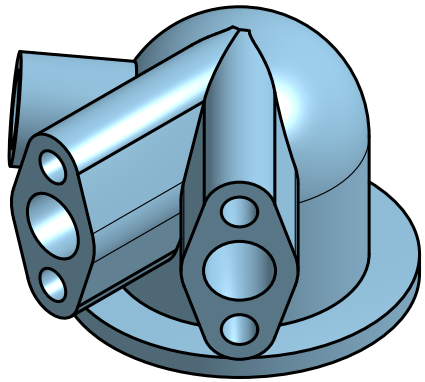
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	04/11/2022
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE		
SIZE A	DWG NO.	REV.
SCALE 1:2	WEIGHT	SHEET 1 of 1

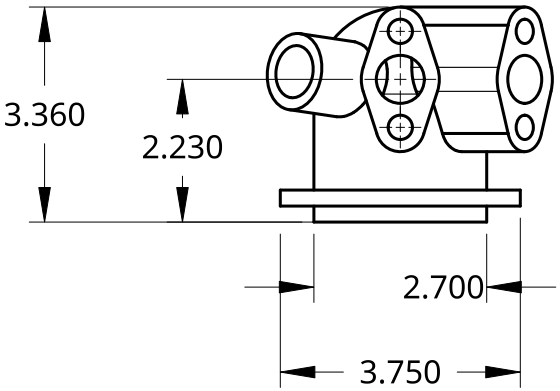


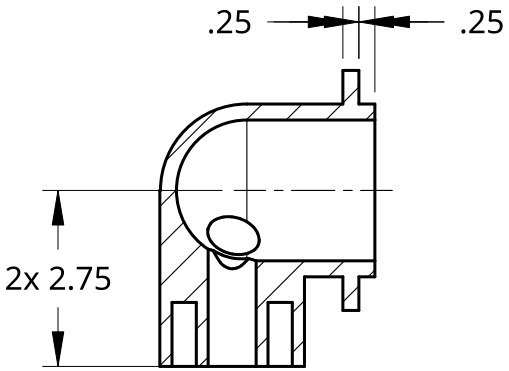


SECTION A - A

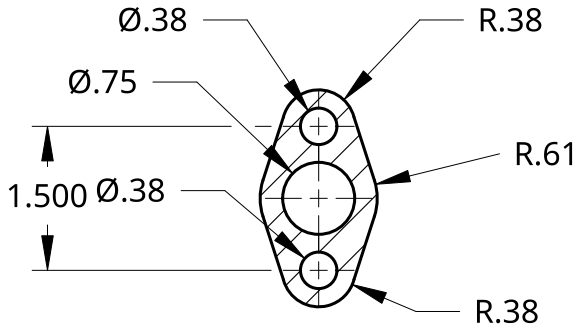


SCALE 1:2





SECTION C - C



SECTION B - B
SCALE 1:2

UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES .XX = ±.0- .XXX = ±.00- ANGULAR = ± ° .XXXX = ±.000- FRACTIONAL = ± SURFACE FINISH		NAME	DATE
	DRAWN	SAMUEL BREAUX	08/17/2024
	CHECKED		
	APPROVED		
DO NOT SCALE DRAWING			
BREAK ALL SHARP EDGES AND REMOVE BURRS			
THIRD ANGLE PROJECTION	MATERIAL	FINISH	

TITLE	Model Mania Practice 2	
SIZE A	DWG NO.	REV.
SCALE 1:3	WEIGHT	SHEET 1 of 1

Glove-Controlled Robotic Arm

Southeastern Louisiana University

Engineering Technology 494-01

December 09, 2024

Participants: Samuel Breaux, Sean Bruce, & Chloe Hill

Coordinator: Dr. Ahmad Fayed

Advisor: Dr. Cris Koutsougeras

Table Of Contents

I. Abstract

This project aims to create an innovative glove-controlled robotic arm designed to bridge the gap between human operators and remote environments. The use of a glove interface intends to act as an intuitive human-machine interface (HMI) for the robotic arm. Through finger-tracking and object-tracking methods, users can control a robotic system in real-time.

II. Introduction

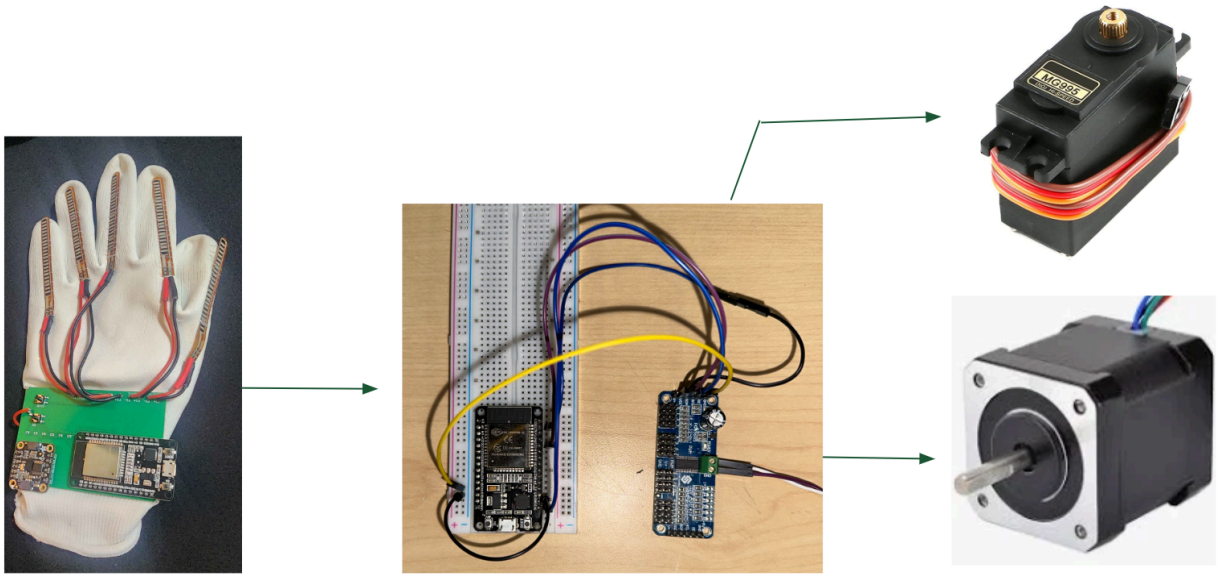


Figure 1: Physical System Overview

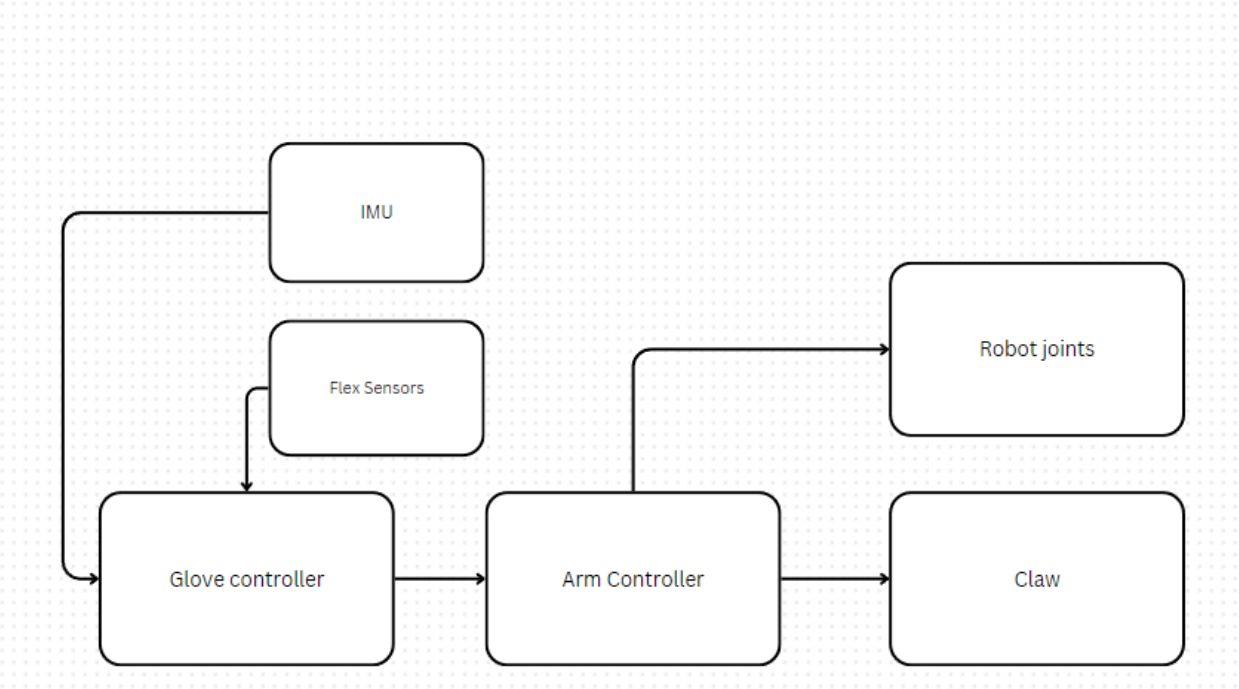


Figure 2: Conceptual System Overview

Over the course of this project, the group created a glove-controlled robotic arm by using various sensors to track data from the hand and then moving the arm based on this data. In the above figures are a physical and conceptual system overview showing how each component feeds into the system. The glove controller is a microcontroller receiving inputs from the inertial measurement unit, flex sensors, and a few buttons for safety and calibration. The robotic arm has a microcontroller, a servo driver, a stepper motor, an H-bridge, and five servo motors. The microcontroller of the robot acts as a receiver of the glove data and then moves the proper joints according to that input. The robotic arm also consists of multiple mechanical components that were designed and 3D printed for this project.

III. Goals

The main goal of this project was to create an intuitive HMI that could control a simple robotic device. To accomplish this, the group made a set of measurable objectives known as deliverables to determine the success of the project. The deliverables can be seen in the following table:

Deliverables
Operator Glove assembly and robot controls system program (Chloe)
3D printing and assembly of robotic arm (Sean)
Operator Glove PCB design and communications program (Samuel)
Robotic arm assembly controlled by an operator glove capable of gripping and releasing an object

Figure 3: Deliverables

By the end of this project, all of the deliverables were successfully completed. The specifics of each subsystem used to achieve the final result will be elaborated on in the system details.

IV. System Details

A. Glove Details

An ESP32 was chosen as the communication device for the glove due to its versatility, power efficiency and robust feature set. It offers both Wi-Fi and bluetooth capabilities, which are essential for seamless wireless communication with other devices. The ESP32's dual-core processor ensures efficient handling of multiple tasks simultaneously, making it ideal for processing data from the flex sensor in real-time while managing communication. Its low power

consumption is particularly valuable for wearable projects. Additionally, the ESP32 is well supported by a large development community, providing access to numerous libraries, tutorials, and resources, which simplifies the integration of sensors and communication protocols. Its small form factor and cost effectiveness make it an ideal choice for compact and reliable wearable technology.

The glove controller uses five flex sensors, five resistors, two buttons, an inertial measurement unit (IMU), and an ESP-WROOM-32 microcontroller. The five flex sensors in combination with the resistors make voltage dividers. The output voltage of these voltage dividers is fed to the GPIO pins of the microcontroller and acts as a signal that represents the angle of flexion of the flex sensor. For clarification, the following figure shows the details of a voltage divider.

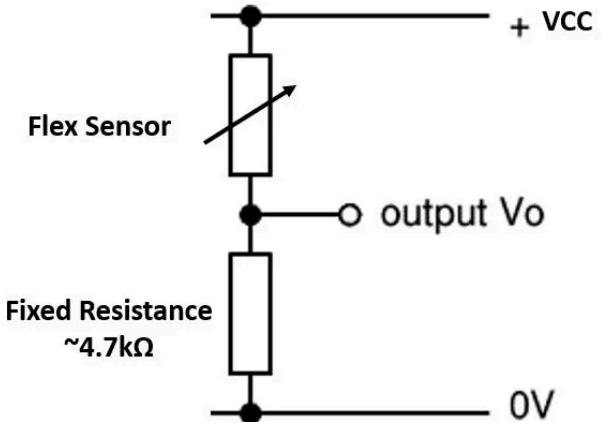


Figure 4: Voltage divider example

The flex sensors are sewn to the glove so that this angle of flexion will correlate to the movements of the operator's hand. The movements produced by these flex sensors were jittery due to noise. To overcome this obstacle, the group made a threshold value in the program and

when this threshold was exceeded the robot would move at a stable speed. This way, the speed wasn't fluctuating based on the noise. However, given more time, an impedance buffer could be added to the pcb design to reduce noise in the flex sensors. PID control could also be used to smoothly move between different speeds for more dynamic and fluid control. The IMU is responsible for keeping track of the hand orientation and communicates with the microcontroller using I^2C. The specific IMU used in this model was the BNO085 by adafruit shown in Figure 5.

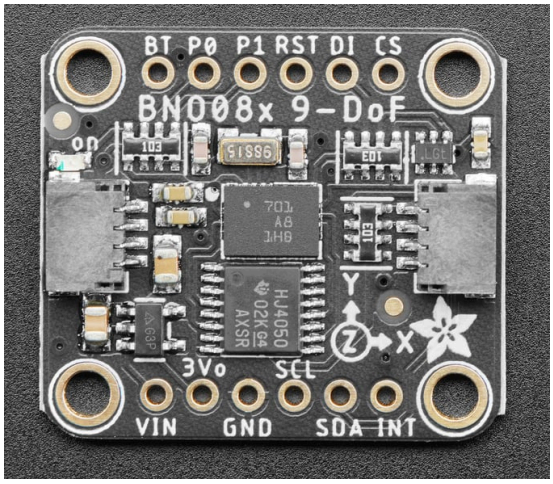


Figure 5: BNO085 IMU

The only other components used in this design were two simple push buttons for emergency stopping of the robot as well as a calibration button. All of these components were connected via a printed circuit board designed by the group. The circuit schematic can be seen in the following figure.

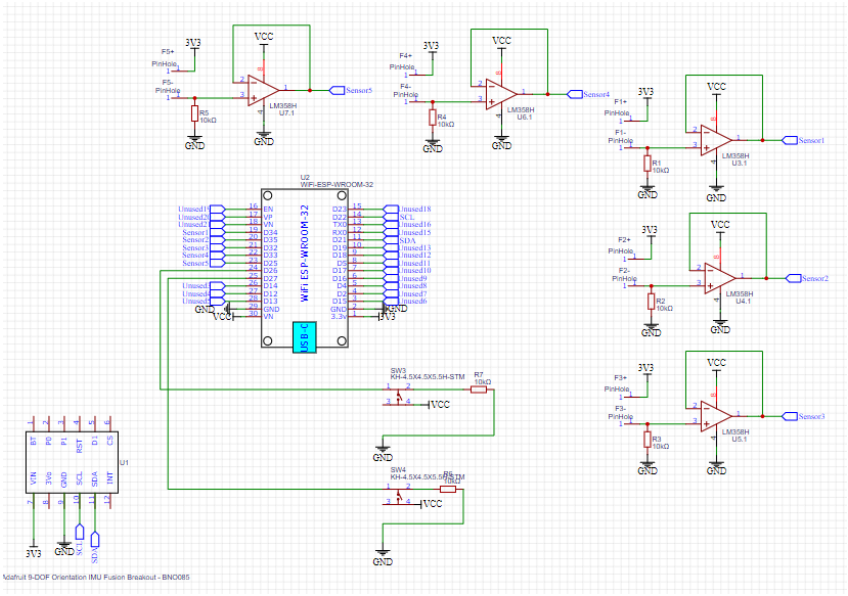


Figure 6: Glove Controller PCB Schematic

All of the sensors ultimately feed their data back to the ESP-WROOM-32 which has a program made to send out this information over ESP Now in a data structure consisting of floats for the flex sensors and angle data, and booleans for the buttons. The program can be seen in appendix A. This program is responsible for making the glove act as the sender of the two microcontrollers used in this system and its only job is to broadcast the data constantly being controlled by the ESP-WROOM-32. The methods of using this data to control the robot are found in the next section. The final assembly of the glove can be seen below in Figure 6 and acted as a successful robot controller in our project.

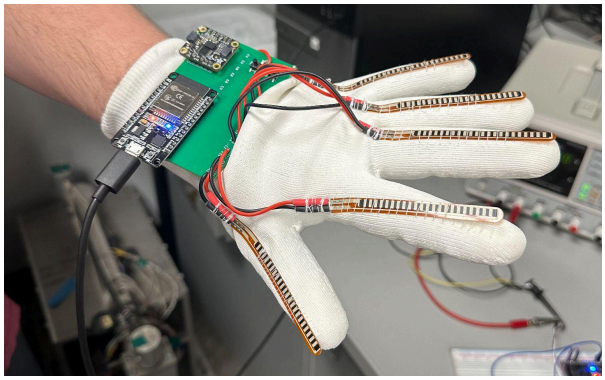


Figure 7: Final Glove Assembly

B. Arm Details

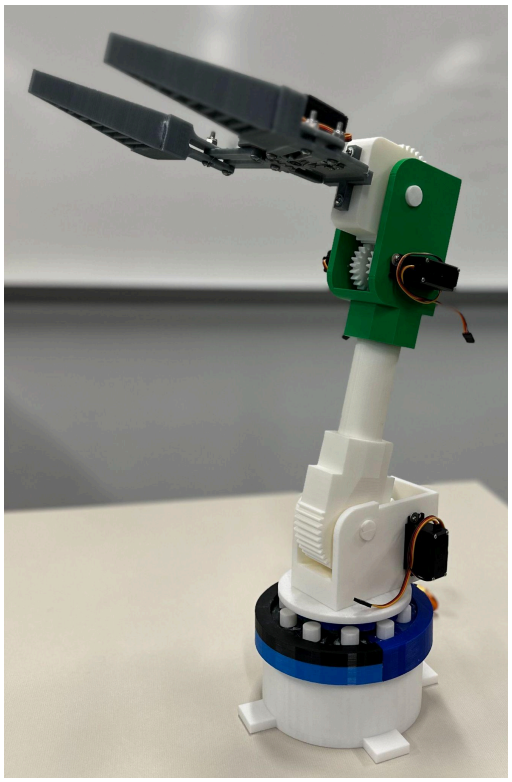


Figure 8: Assembled Robotic Arm

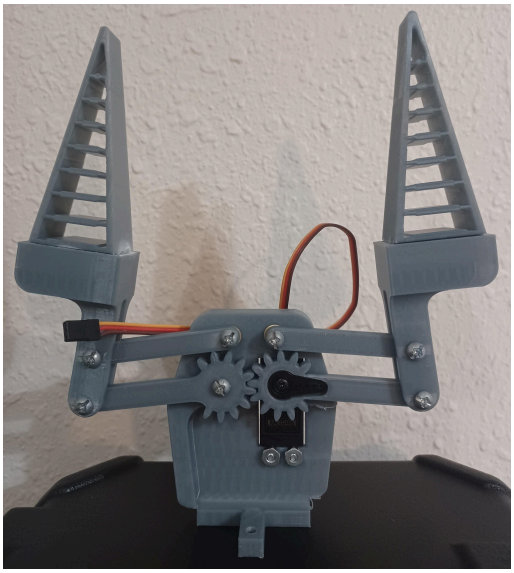


Figure 9: End-of-arm Tooling

The body of the robotic arm (Figure 8) is 3D printed and includes three degrees of freedom (joints) and an end-of-arm tooling. The end-of-arm tooling, the “claw” (Figure 9), has a body with a cross-style base with two holes for mounting. Higher on the right

side of the body is a hole for an MG995 motor to be mounted face-up. A geared beam is attached to the servo head and supported by another beam above it. These beams control the motion of the right claw. This two-beam design is mirrored to the left claw to complete the scissoring motion that allows for gripping objects.

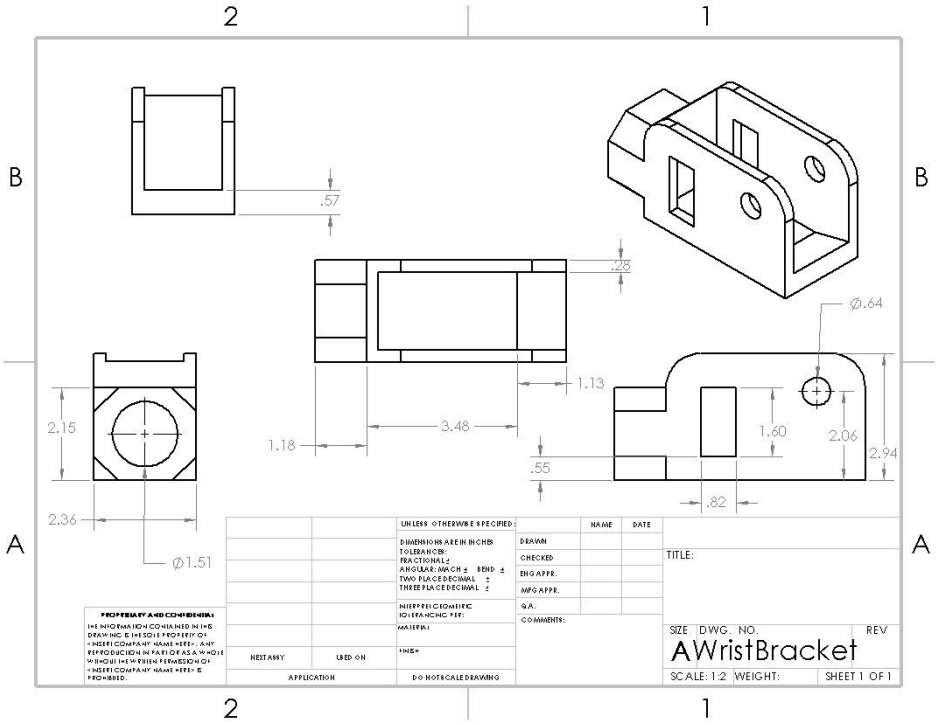


Figure 10: Wrist Bracket

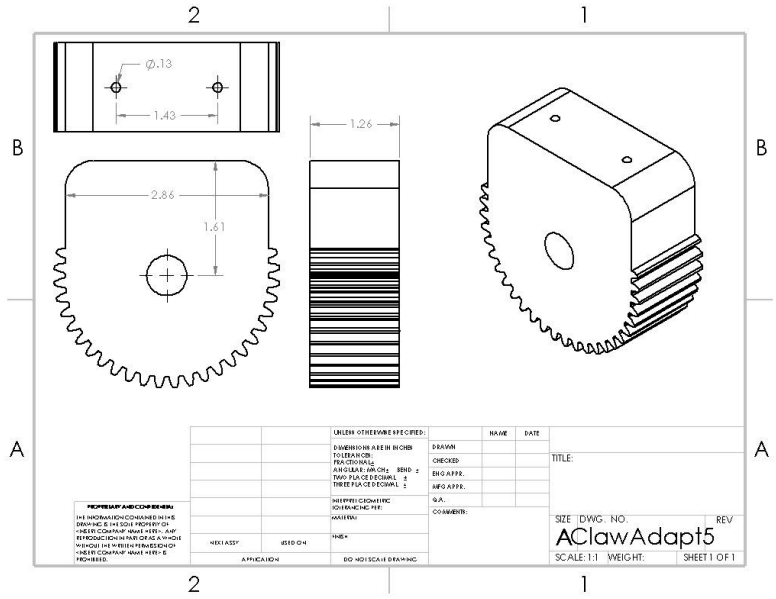


Figure 11: Claw Mount Gear

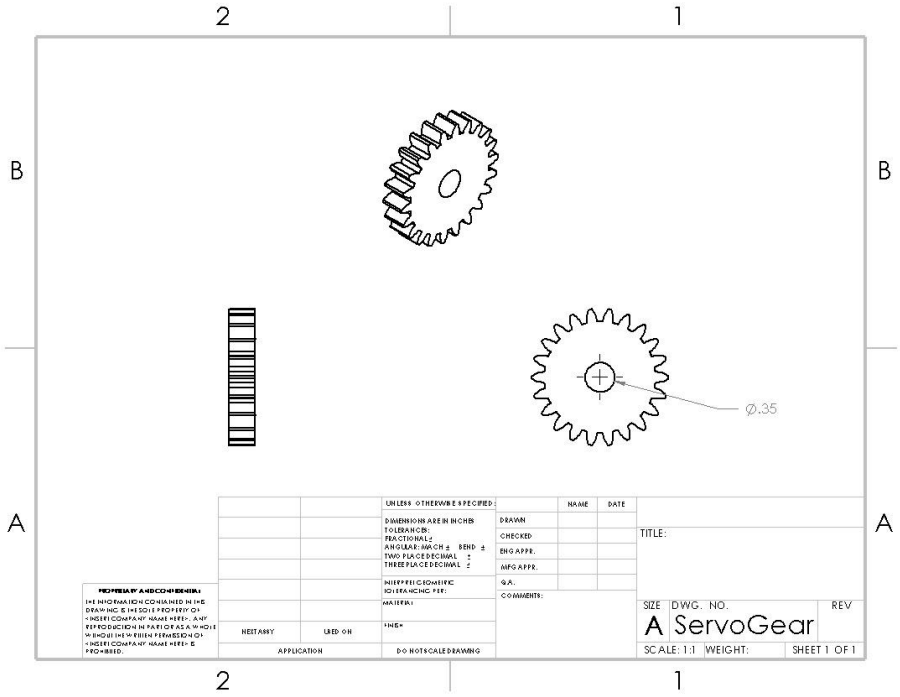


Figure 12: Servo Gear

The “wrist” joint begins with mounting the claw to the claw mount gear (Figure 11). This gear is driven with a two-to-one gear ratio from two servo gears (Figure 12)

attached to the MG995 servos. Two of these servos are mounted facing each other in the wrist bracket (Figure 10). By inverting the input controls of one servo and syncing their output, both motors move together in the same direction to produce double the torque while driving the bigger gear.

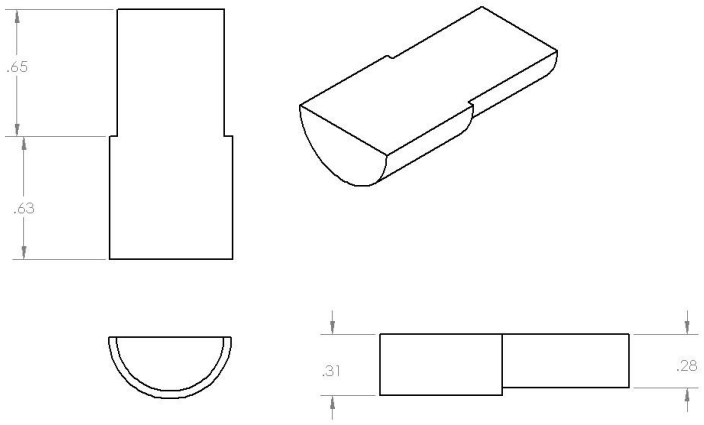


Figure 13: Arm Pin Half

The claw mount gear is held in place by two arm pins (Figure 13). The pins are 3D printed in halves with the flat face down so that the part is stronger on the z-axis. After gluing the two halves, the pins can be easily inserted and removed for any troubleshooting or testing of the individual assemblies. Otherwise, the pins hold the claw mount gear relatively centered while allowing just enough tolerance to rotate freely.

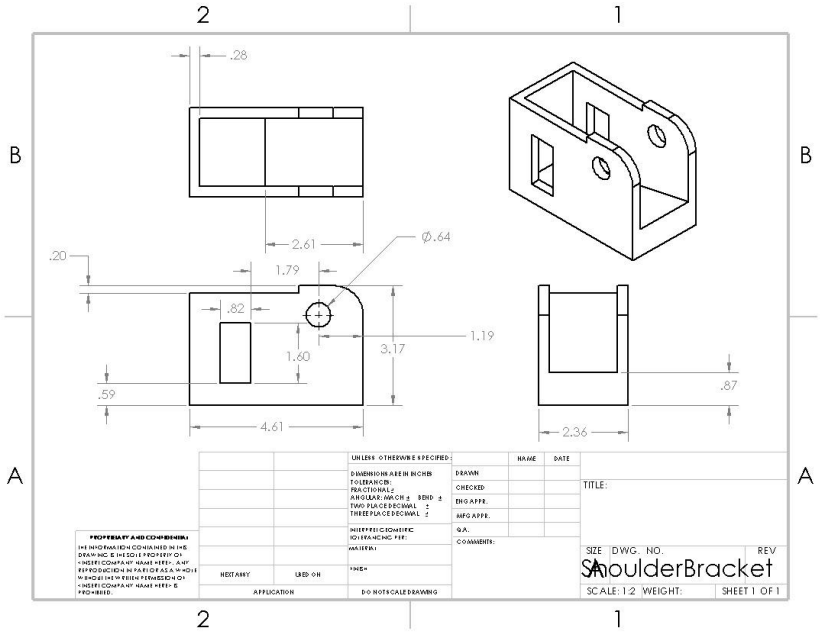


Figure 14: Shoulder Mount Bracket

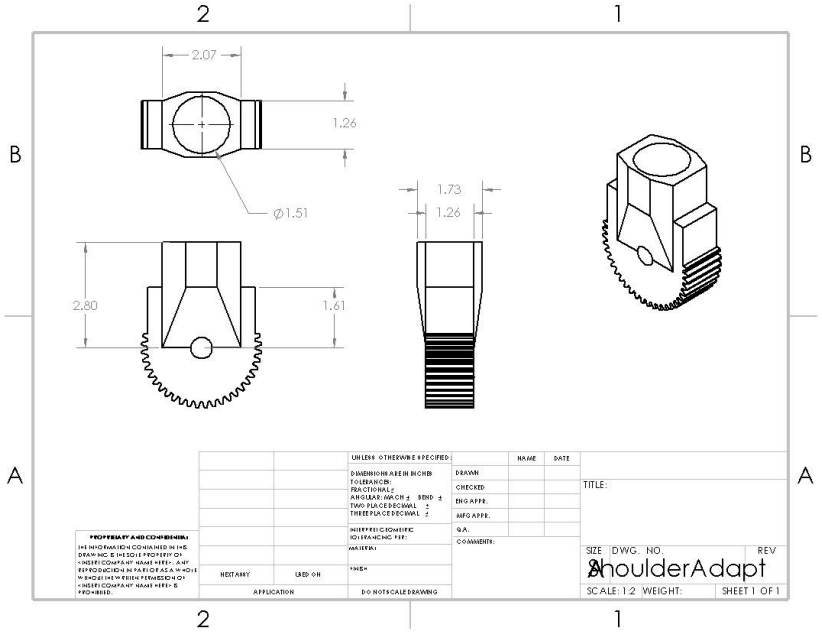


Figure 15: Elbow Gear

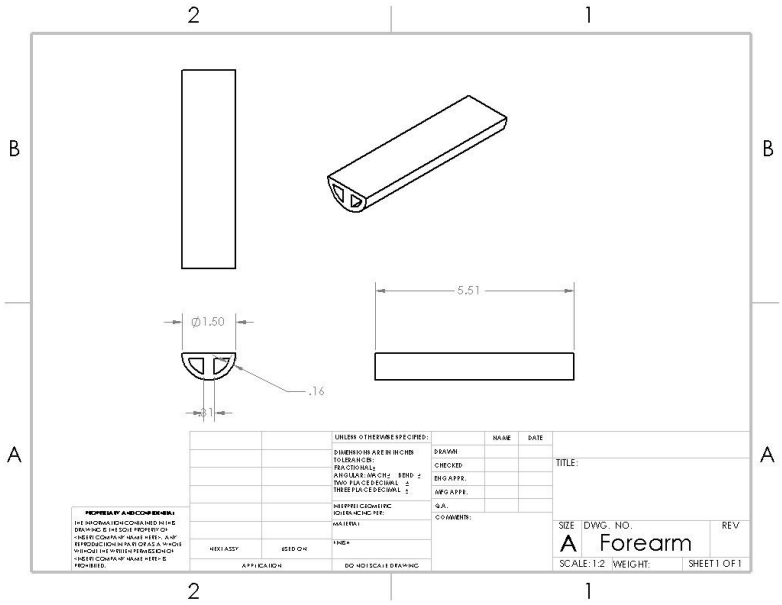


Figure 16: Forearm Half

A similar design is implemented for the arm's "elbow" joint. The forearm halves (Figure 16) are printed and assembled the same as the pins to increase z-axis strength. Once whole, the forearm connects the elbow gear (Figure 15), mounted to the shoulder mount bracket (Figure 14) by two more arm pins, to the wrist bracket. Then, through a two-to-one gear ratio to two servos, the arm's length is manipulated.

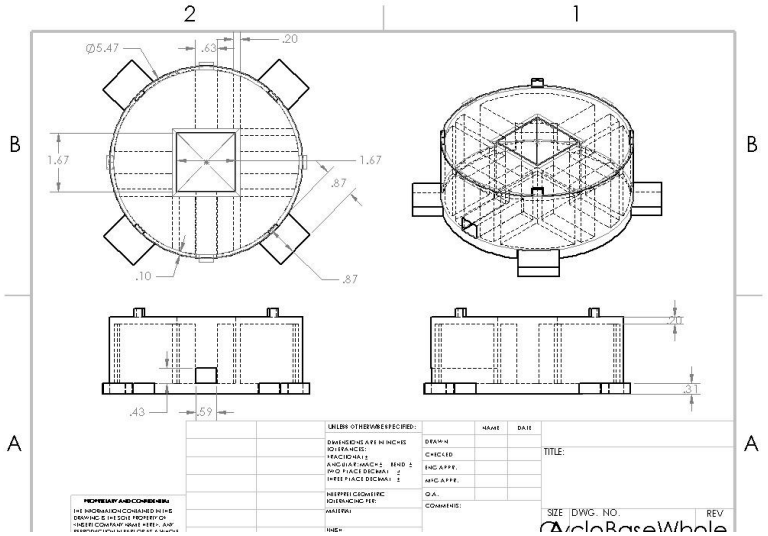


Figure 17: Arm Base

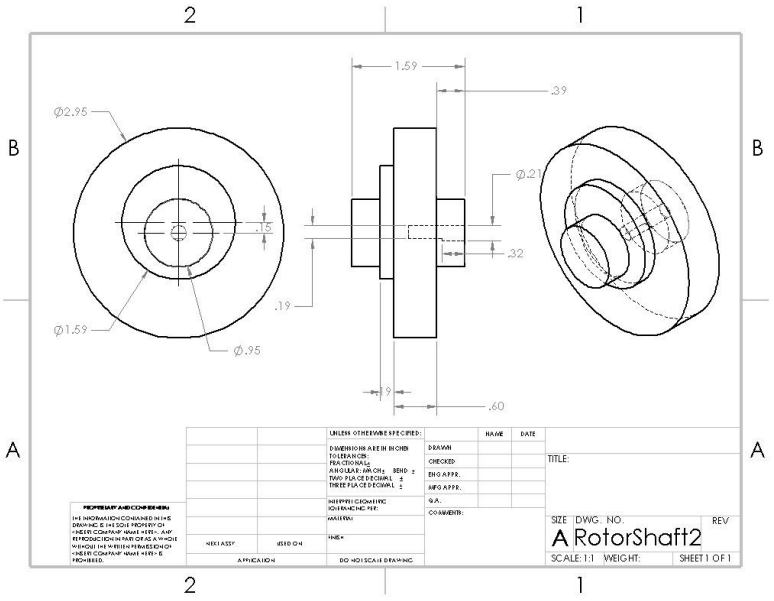


Figure 18: Input Shaft

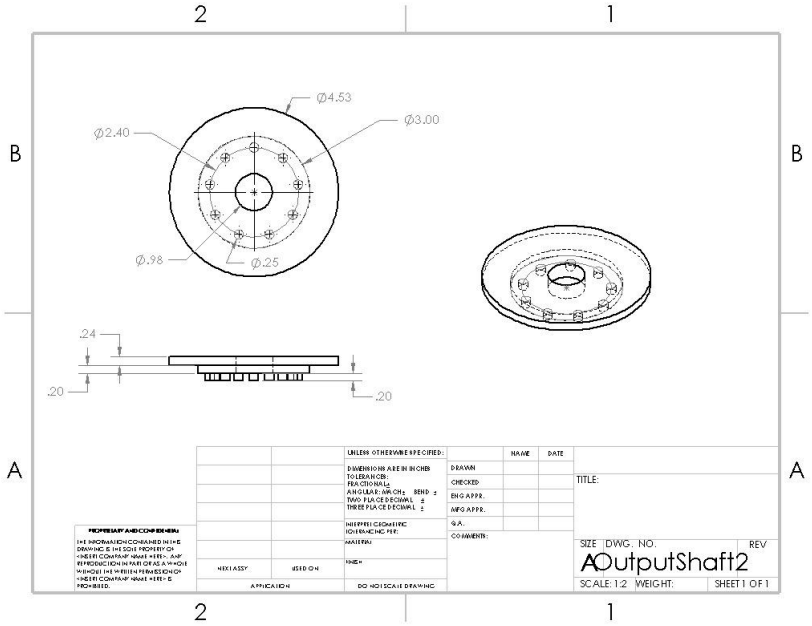


Figure 19: Output Disk

The "shoulder" joint, or base, is manipulated by a Nema 17 Bipolar 2A stepper motor that rotates the input shaft to a cycloidal gear drive. This gear drive is a modified recreation of a tutorial in the blog post *Building a Cycloidal Drive with SOLIDWORKS* by Omar Youmis. The design's size is decreased to twenty percent of the original to make it more relative to the size of the arm. The input shaft (Figure 18) includes a full disk for better support of the arm as well as a shorter shaft and a key hole to insert the shaft of the stepper motor. The output disk (Figure 19) has an increased diameter to also better the arm's support. The gear drive's high gear reduction ratio allows for a high torque output to rotate the arm at an efficient rate. The shoulder mount bracket is mounted off-center on the output disk so the weight of the arm is more towards the center of the gear drive. Then, the entire gear drive is mounted on the arm base (Figure 17). The arm base is printed to support the weight of the arm vertically and horizontally with a square hole in the center for the stepper motor to sit in and a small channel at the bottom for the motor's

wires. Though, after its printing, the base had to be modified. The four small tabs atop the base needed removal, and one wall of the center hole needed slight adjustment because the wires were unable to be inserted properly during the insertion of the stepper motor.

To control and power the arm, the arm's ESP32 provides 3.3 Volts to and communicates with a Sunfounder PCA9685 servo driver. The 3.3 Volts powers the logic of the driver while an external 6 Volt power supply is wired in to provide power for the servos. As the ESP32 receives input, it translates it to the servo driver, and the driver directs the proper current as needed. Additionally, the ESP32 controls the stepper motor using an L293DNE H-bridge chip. The chip uses 5 Volts on the Vcc1 pin to manipulate the logic signals while 12 Volts is supplied to the Vcc2 pin to power the motor.

V. Division of Works

Samuel Breaux	Sean Bruce	Chloe Hill
● Research and design Glove Hardware ● Assist in programming the glove and arm	● Mechanics, movement, and design of the arm and claw	● Making codes for arm and glove coordination ● Assist with wiring arm

VI. Challenges

For research and design of the glove hardware, the main challenge has been in deciding which method to use for finger tracking. Last semester, the progress for the flex sensor glove took longer than expected so the team decided to explore other options. The main benefit of the other option our team is researching, which is using an exoskeleton with potentiometers

measuring the angle joints, is that it offers higher precision than the flex sensor version. This is because the flex sensor design inherently has more variance in the resistance from small or insignificant movements. However, the exoskeleton design requires 3D printing which can be challenging with multiple small parts fitting together. Any warping or deformation of the parts can cause them to be unusable especially when parts fitting together have fairly low tolerances. This was especially noticeable in the claw design last semester. The 3d printing process for the arm has also been difficult throughout the entire process, often shifting the timeline to accommodate changes made to the design.

One of the main challenges in coding the glove was identifying the right libraries to use in Arduino IDE. For reasons that weren't immediately clear, some libraries worked only on certain computers, which caused a lot of confusion and led to back-and-forth troubleshooting. This made it difficult to determine which libraries were compatible and reliable across different setups.

Soldering wires to a PCB presented several challenges, especially for beginners. One common difficulty was ensuring that the soldering iron was at the correct temperature. The soldering iron that was used did not specify if it was in Celsius or Fahrenheit so it was a guessing game. Another challenge was making sure that the joints were solid and clean so that they provided a good connection. Once the soldering was done from the PCB to the flex sensor they then needed to be sewn to the glove which presented more challenges. The flex sensors did not stay securely in place in their slits so many different methods were used to get them to not come undone. While sewing directly through the flex sensor seemed like an option it was avoided to prevent potential damage to the delicate components.

The mechanics and movement of the arm are a challenge in their own right. The translation of power between motors has to be accurate and efficient to properly translate the data received from the glove as well as function as a whole. If one motor is implemented incorrectly, it will affect the stability and function of the other joints. Every motor has to support and move the weight of its relative joint, the joints and structure above it, and the payload putting strain on the entire arm.

Near the project's conclusion, the group had issues with the shoulder joint. The stepper motor was unable to turn the output disk when weight was applied. This is believed to be because of lack of support to the output disk. When weight was present, the motor’s shaft was liable to experience a load that was unconventional to its design. Due to lack of time, the stepper motor, along with the cycloidal gear drive, had to be removed from the final assembly. Also, the arm base could not prevent the arm from tipping over at full extension. To solve the output disk issue, the outer walls of the arm base can be raised up to the height of the disk, and an added lip with decent supports can allow the disk to rest so the weight is more on the base. As to solve the tipping issue, the arm base’s design can be widened for better weight distribution.

VII. Materials

Robotic Arm	
Quantity	Item
5	MG995 Servo Motor
1	Nema 17 Bipolar Stepper Motor (17HS19-2004S1)
1	SunFounder PCA9685 Servo Driver
1	6 Volt Power Supply
1	ESP-WROOM-32
1	L293DNE Quadruple H-Bridge
1	Breadboard
As Needed	Jumper Wires

Control Glove	
Quantity	Item
1	Fabric Glove
1	Custom PCB
1	BNO085 Inertial Measure Unit
5	Flex Sensor
1	ESP-WROOM-32
10	Wires

Appendix A: Glove Controller Program Code

```
#include <Adafruit_BNO08x.h> // Import for IMU Module
#include <esp_now.h>
#include <WiFi.h>

// Pin Definitions
#define BNO08X_CS 10
#define BNO08X_INT 9
#define BNO08X_RESET -1
#define ERButton 24
#define calibButton 25

// IMU and ESP-Now Variables
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
Adafruit_BNO08x bno08x(BNO08X_RESET);
sh2_SensorValue_t sensorValue;

// Sensor Pins and Data Vectors
int sensorIDVector[5] = {34, 35, 32, 33, 25}; // P1–P5
float sensorDataVector[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

// Flags and Configurations
bool ENABLE = true;
bool ER = false;
bool calibTime = false;

// Struct to Hold Data
typedef struct {
  bool calibTime;
  bool ER;
  float dataVector[9];
} data_vector;

data_vector data;
esp_now_peer_info_t peerInfo;

// Callback Function for ESP-Now
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

// Function to Initialize Sensor Reports
void setReports() {
  Serial.println("Setting desired reports");
```

```
  if (!bno08x.enableReport(SH2_GAME_ROTATION_VECTOR)) {
    Serial.println("Could not enable game vector");
  }
}

// Function to Update Sensor Data
void updateSensorData(int IDVector[5]) {
  // Handle Calibration Time
  if (data.calibTime) {
    delay(5000); // Wait for calibration and button release
  }

  // Toggle Emergency Reset
  if (digitalRead(ERButton)) {
    data.ER = !data.ER;
    delay(500); // Debounce delay
  }

  // Update Calibration and Analog Sensor Data
  data.calibTime = digitalRead(calibButton);
  for (int i = 0; i < 5; i++) {
    data.dataVector[i] = map(analogRead(IDVector[i]), 0, 4095, 0, 265);
  }

  // Retrieve IMU Sensor Data
  if (bno08x.getSensorEvent(&sensorValue)) {
    if (sensorValue.sensorId == SH2_GAME_ROTATION_VECTOR) {
      data.dataVector[5] = sensorValue.un.gameRotationVector.real;
      data.dataVector[6] = sensorValue.un.gameRotationVector.i;
      data.dataVector[7] = sensorValue.un.gameRotationVector.j;
      data.dataVector[8] = sensorValue.un.gameRotationVector.k;

      // Debug Output
      Serial.print("Game Rotation Vector - r: ");
      Serial.print(data.dataVector[5]);
      Serial.print(", i: ");
      Serial.print(data.dataVector[6]);
      Serial.print(", j: ");
      Serial.print(data.dataVector[7]);
      Serial.print(", k: ");
      Serial.println(data.dataVector[8]);
    }
  }
}

// Function to Transmit Data via ESP-Now
void sendSensorData() {
```

```
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&data, sizeof(data));
  if (result == ESP_OK) {
    Serial.println("Sent with success");
  } else {
    Serial.println("Error sending the data");
  }
  delay(50);
}

// Arduino Setup Function
void setup() {
  pinMode(calibButton, INPUT);
  pinMode(ERButton, INPUT);
  Serial.begin(115200);

  // Initialize ESP-Now
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-Now");
    return;
  }
  esp_now_register_send_cb(OnDataSent);

  // Register Peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  // Initialize IMU
  if (!bno08x.begin_I2C()) {
    Serial.println("Failed to find BNO08x chip");
    while (1) {
      delay(10);
    }
  }

  Serial.println("Setup Complete");
}

void loop() {
  if (bno08x.wasReset()) {
    Serial.print("Sensor was reset. ");
```

```
    setReports();
  }

  if (ENABLE) {
    updateSensorData(sensorIDVector);
    sendSensorData();
  }
}
```

Appendix B: Arm Controller Program Code

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

#define SERVOMIN  90 // Minimum Servo
#define SERVOMAX  605 // Maximum Servo PWM Value
int motorFinger[4] = {9, 1, 2, 4};
int motorAngles[3] = {0, 0, 0}; // Angle of each motor or pair of motors
float lowerCalibs[4] = {0, 0, 0, 0}; // Lower calibration values. Doesn't need initial values
float higherCalibs[4]; // Higher calibration values
bool calibrated = false; // Calibration flag
bool moving = false; // Moving flag
bool moveSynchro1; // Moving flag for joint 1
bool moveSynchro2; // Moving flag for joint 2
bool moveClawMotor; // Moving flag for joint 3
bool ER; // Emergency Stop
bool calibTime; // Used for recalibration

typedef struct data_vector {
 bool calibTime; // Used for recalibration
 bool ER; // Emergency Stop
 float dataVector[9];
} data_vector;

data_vector data; // Data vector received from the glove
float receivedData[9]; // Array for storing float data received from the glove

// Callback function executed when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len) {
 int directionNumIndex = 7; // Index of IMU data with axis used to determine the direction of joint movements
 memcpy(&data, incomingData, sizeof(data));
 ER = data.ER;
 calibTime = data.calibTime;

 // Update the receivedData array
 for (int i = 0; i < 9; i++) {
  receivedData[i] = data.dataVector[i];
  if (i == 0) receivedData[i] = 0;

  // Maps received data according to proper calibration values per finger
  if (calibrated && i < 4) {
   receivedData[i] = map(receivedData[i], lowerCalibs[i], higherCalibs[i], 0, 100);
   receivedData[i] = constrain(receivedData[i], 0, 100);
  }
```

```
  }

  if (calibrated) {
    int aboveThresholdCount = 0;
    int largestIndex = 0;
    int largestValue = 0;
    for (int i = 0; i < 4; i++) {
      if (receivedData[i] > largestValue) {
        largestValue = receivedData[i];
        largestIndex = i;
      }
      if (receivedData[i] > 60) aboveThresholdCount++;
    }

    if (aboveThresholdCount > 1) {
      updateMotors(largestIndex, directionNumIndex);
    } else {
      for (int i = 0; i < 4; i++) updateMotors(i, directionNumIndex);
    }
  }
}

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

// Synchronizes two motors
int motorSync(int drivingMotor, int drivenMotor, int direction, int currentAngle, int speed = 5) {
  if (direction == 1) {
    currentAngle = constrain(currentAngle + speed, 5, 175);
    pwm.setPWM(drivingMotor, 0, angleToPulse(currentAngle));
    pwm.setPWM(drivenMotor, 0, angleToPulse(180 - currentAngle));
  } else {
    currentAngle = constrain(currentAngle - speed, 5, 175);
    pwm.setPWM(drivingMotor, 0, angleToPulse(currentAngle));
    pwm.setPWM(drivenMotor, 0, angleToPulse(180 - currentAngle));
  }
  delay(50);
  return currentAngle;
}

// Smooth motor movement
int motorSmooth(int motor, int direction, int currentAngle, int speed = 10) {
  if (direction == 1) {
    currentAngle = constrain(currentAngle + speed, 5, 175);
    pwm.setPWM(motor, 0, angleToPulse(currentAngle));
  } else {
    currentAngle = constrain(currentAngle - speed, 5, 175);
    pwm.setPWM(motor, 0, angleToPulse(currentAngle));
```

```
  }
  delay(50);
  return currentAngle;
}

// Updates motor values based on received data
void updateMotors(int finger, int directionNumIndex) {
  int triggerThreshold = 60;
  if (receivedData[finger] > triggerThreshold) {
    if (finger == 1) {
      moveSynchro1 = true;
      moveSynchro2 = false;
      moveClawMotor = false;
    } else if (finger == 2) {
      moveSynchro2 = true;
      moveSynchro1 = false;
      moveClawMotor = false;
    } else if (finger == 3) {
      moveClawMotor = true;
      moveSynchro1 = false;
      moveSynchro2 = false;
    }
  } else {
    moveClawMotor = false;
    moveSynchro1 = false;
    moveSynchro2 = false;
  }
}

// Calibrates flex sensor values for each finger
void calibration() {
  Serial.println("Hold hand open");
  delay(500);
  Serial.println("3");
  delay(1000);
  Serial.println("2");
  delay(1000);
  Serial.println("1");
  delay(1000);

  for (int i = 0; i < 4; i++) {
    lowerCalibs[i] = receivedData[i];
  }

  Serial.println("Hold hand closed");
  delay(500);
  Serial.println("3");
```

```
 delay(1000);
 Serial.println("2");
 delay(1000);
 Serial.println("1");
 delay(1000);

 for (int i = 0; i < 4; i++) {
  higherCalibs[i] = receivedData[i];
 }

 calibrated = true;
}

// Converts angle to PWM signal
int angleToPulse(int angle) {
 return map(angle, 0, 180, SERVOMIN, SERVOMAX);
}

void setup() {
 Serial.begin(115200);
 pwm.begin();
 pwm.setPWMFreq(60);

 pwm.setPWM(1, 0, angleToPulse(5));
 pwm.setPWM(0, 0, angleToPulse(175));
 pwm.setPWM(3, 0, angleToPulse(175));
 pwm.setPWM(2, 0, angleToPulse(5));
 pwm.setPWM(4, 0, angleToPulse(0));

 WiFi.mode(WIFI_STA);

 if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
 }

 esp_now_register_recv_cb(OnDataRecv);
 calibration();
}

void loop() {
 int direction = (receivedData[6] >= 0.6) ? 1 : 0;
 Serial.println(direction);

 if (calibTime) calibration();
 if (ER) {
  Serial.println("Robot will not move until ER is reset");
```

```
  } else if (moveSynchro1) {
    motorAngles[0] = motorSync(1, 0, direction, motorAngles[0]);
  } else if (moveSynchro2) {
    motorAngles[1] = motorSync(3, 2, direction, motorAngles[1]);
  } else if (moveClawMotor) {
    motorAngles[2] = motorSmooth(4, direction, motorAngles[2]);
  }
}
```

References

Gass, Daniel. “Robot Arm Automation.” *Arduino Project Hub*, 7 January 2022, https://projecthub.arduino.cc/danielgass/robot-arm-automation-c4e0cb. Accessed 28 February 2024.

McCurdy, Iain. “DIY Bend Sensor.” *Iain McCurdy*, 2016, http://iainmccurdy.org/diy/bendsensor/bendsensor.html. Accessed 28 February 2024.

Mouser Electronics. *Mouser Electronics*, https://www.mouser.com/ProductDetail/Amphenol-Piher/PT15NH06-202A2020?qs=%2FzPWuUUsT7MKz1MDcCOWCQ%3D%3D. Accessed May 2024.

Mouser Electronics. “PT15NH06-202A2020.” *Mouser Electronics*, https://www.mouser.com/ProductDetail/Amphenol-Piher/PT15NH06-202A2020?qs=%2FzPWuUUsT7MKz1MDcCOWCQ%3D%3D. Accessed 6 May 2024.

3DPrintIt. “3D Printed Powered Exoskeleton Hands (Upgrade v1).” *Thingiverse*, https://www.thingiverse.com/thing:952267. Accessed May 2024.

Toglefritz. “Build a Giant 3D Printed Robot Arm : 83 Steps (with Pictures).” *Instructables*, 2017, https://www.instructables.com/Build-a-Giant-3D-Printed-Robot-Arm/. Accessed 28 February 2024.

Wu, Changcheng et al. “Development of a Low-Cost Wearable Data Glove for Capturing Finger Joint Angles.” *NCBI*, 30 June 2021, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8304804/. Accessed 28 February 2024.

Younis, Omar. “Building a Cycloidal Drive with SOLIDWORKS.” 2014. *Dassault Systèmes*, https://blogs.solidworks.com/teacher/2014/07/building-a-cycloidal-drive-with-solidworks.html.