Armikochan:

A 5-DOF Articulated Manipulator

Armikochan: Where Articulation Meets Innovation!

Contents:

INTRODUCTION

Introducing Armikochan, a cutting-edge 5-DOF (Degrees of Freedom) articulated manipulator that redefines the landscape of robotic arms. Comprising a main body and a similarly crafted smaller replica, this innovative robotic arm offers a unique fusion of precision, versatility, and remote control capabilities. The seamless synchronization between the main body and its scaled replica proportional rotation, making Armikochan a promising prototype for remote surgery machines.

But that's not all! Armikochan is really good at remembering how it moves. So, if it does something once, it can do it again exactly the same way. This makes it perfect for doing jobs in factories and other places where things need to be done just repetitively. Armikochan is like a super smart robot friend that's always ready to lend a hand!

Features

Remote Control Precision:

- Controlled through a small-scale controller with high precision.
- Ideal for candidate for prototype in remote surgery machines due to precise control.

Motion Memory:

- Capable of remembering and replicating motions with precision.
- Offers consistency and reliability in executing repetitive tasks.

Mobile Control Capability:

- Can also be controlled using a mobile phone.
- Enhances accessibility and ease of use.

Gyroscope Control Capability:

- Suitable for various industrial and manufacturing tasks.
- Adaptable to hand gestures.

5 Degrees of Freedom (5-DOF):

- Incorporates 5 joints for versatile movement.
- Allows flexibility in performing a variety of tasks.

3D-MODEL

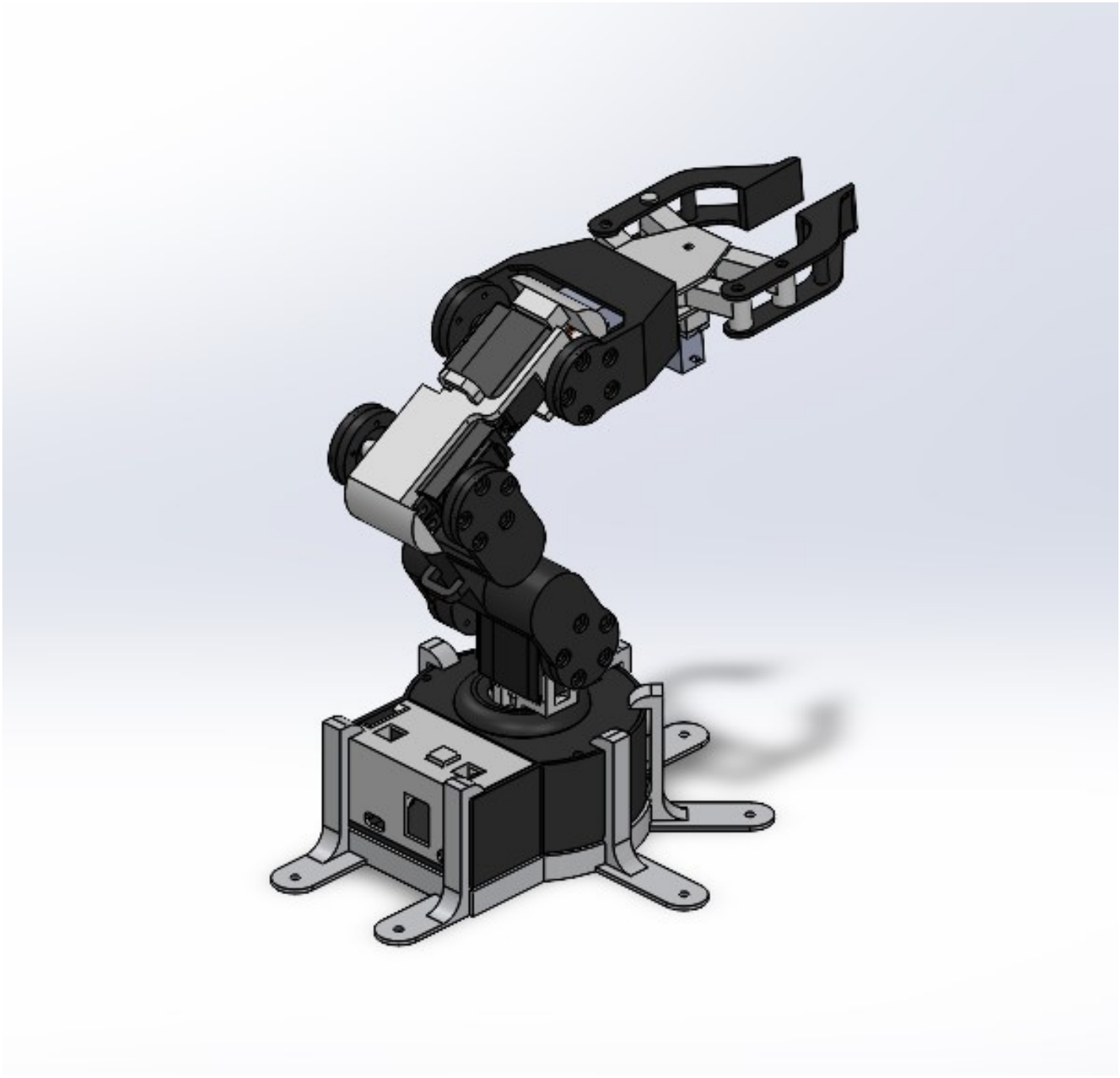


Figure 1: Main Body Adjusted View

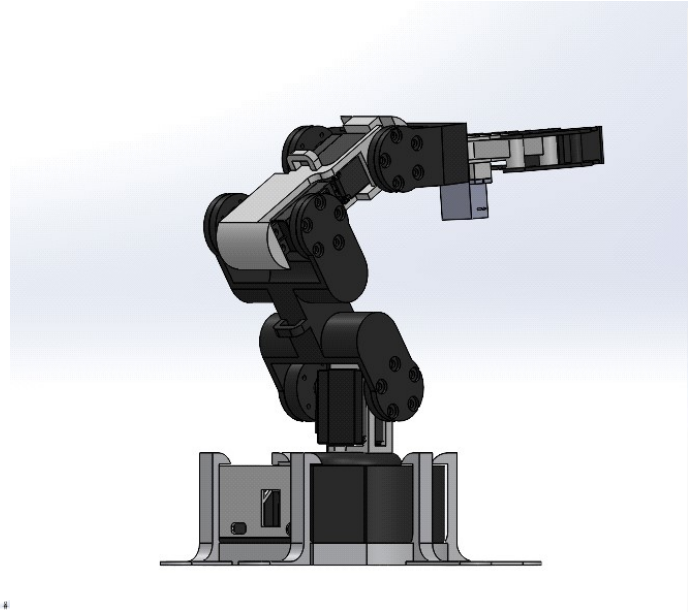


Figure 2: Lateral-view

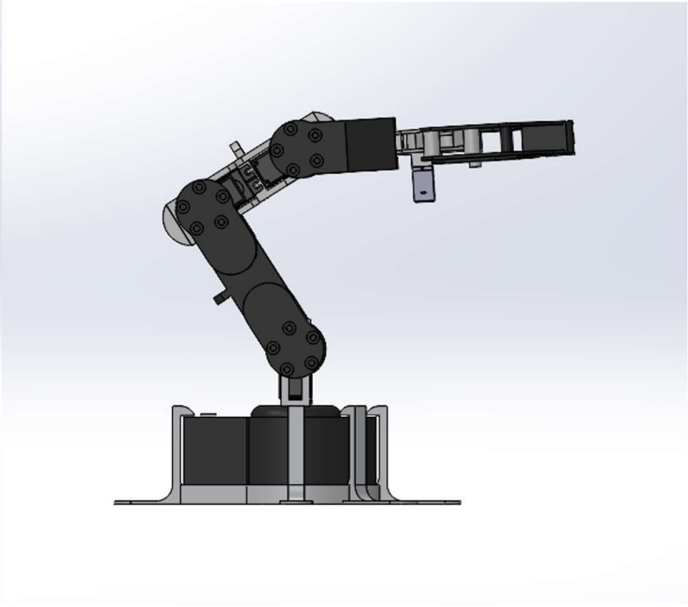


Figure 3: Side View

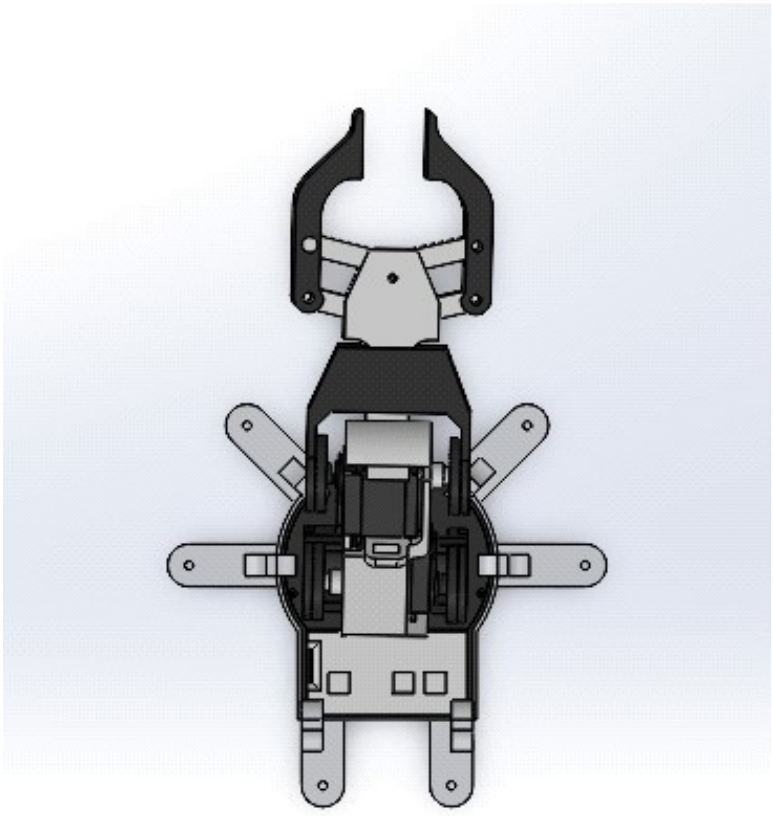


Figure 4: Top View

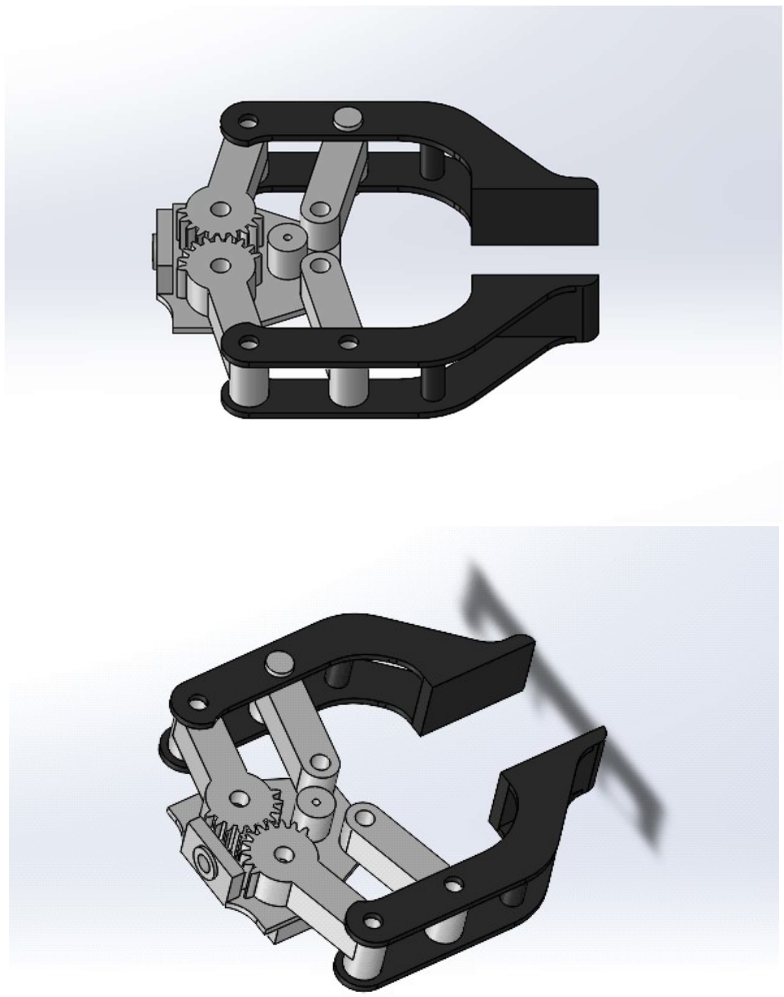


Figure 5: Gripper Mechanism

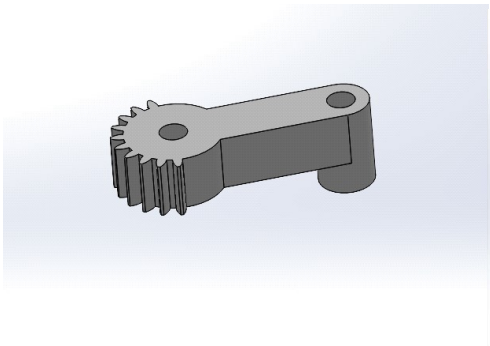


Figure 6: rot-gear

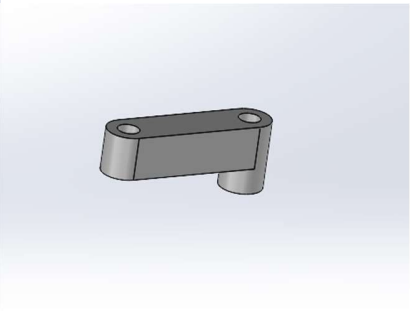


Figure 7: rot-holder

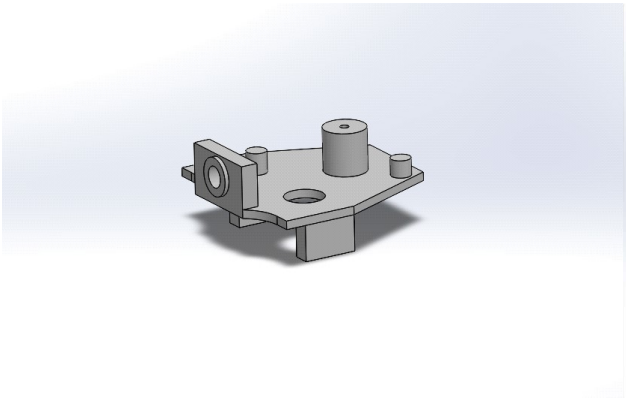


Figure 8: gripper holder

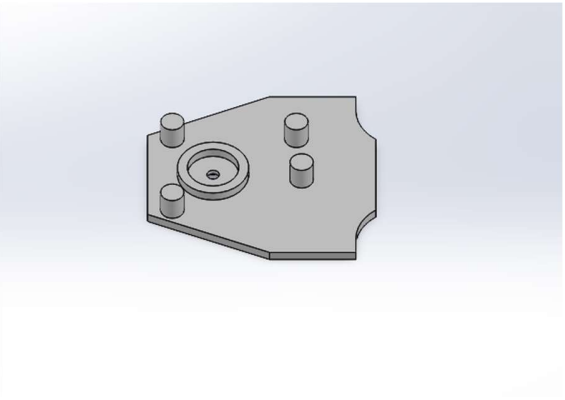


Figure 9: gripper cover

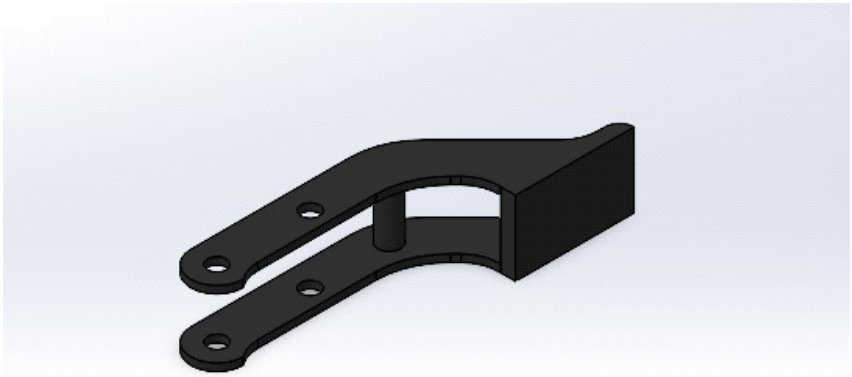


Figure 10: grip tip

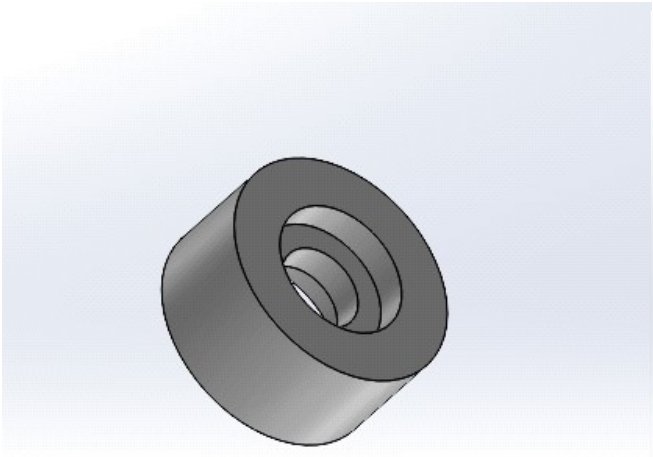


Figure 11: pin holder

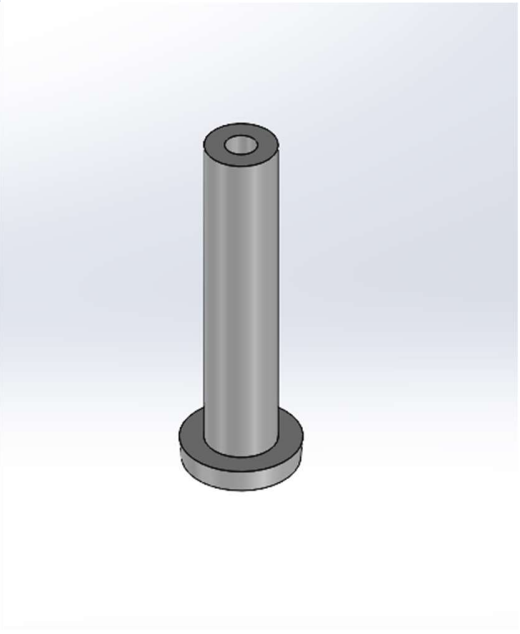


Figure 12: Holder pin

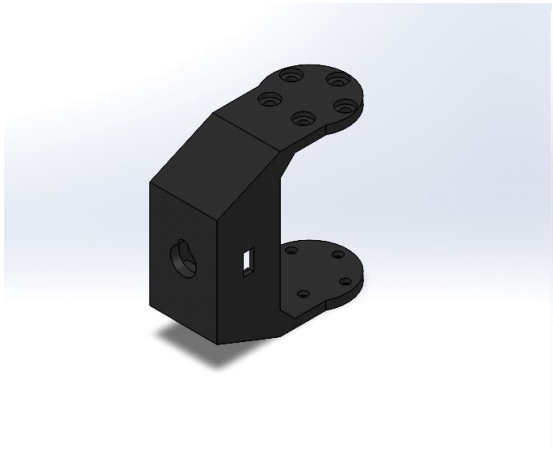


Figure 13: head

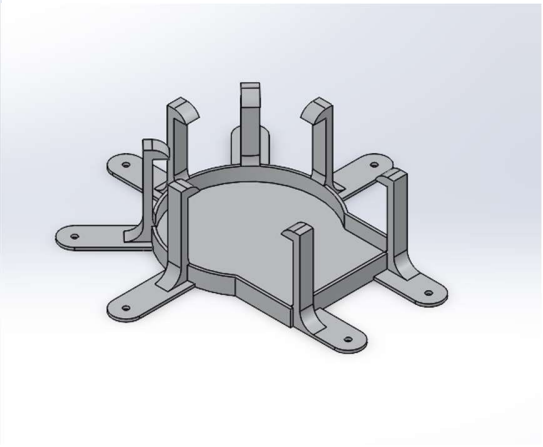


Figure 14: base-body-holder

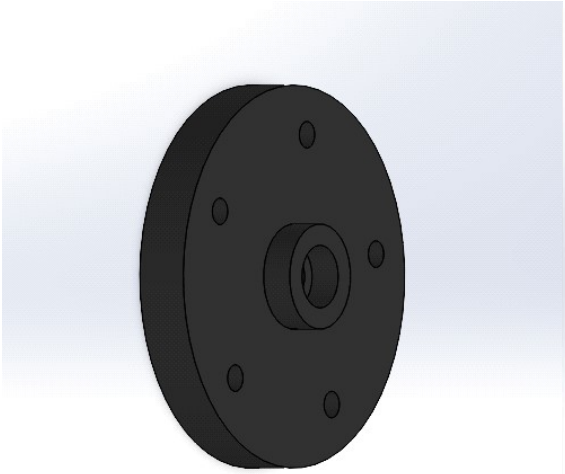


Figure 15: servo horn 1

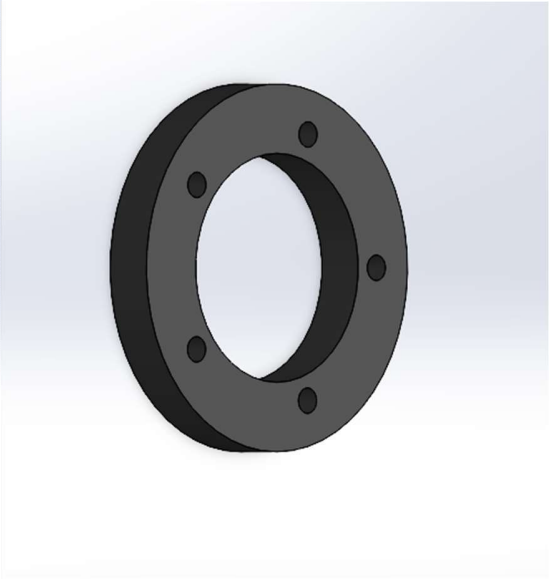


Figure 16: bearing horn

Figure 17: servo holder 1

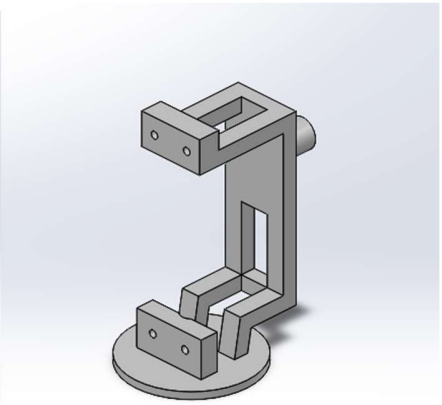


Figure 18: servo horn 2

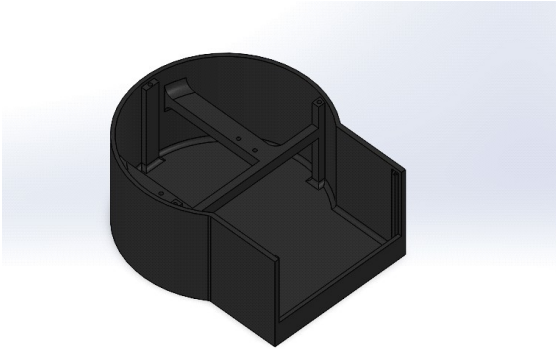


Figure 19: base

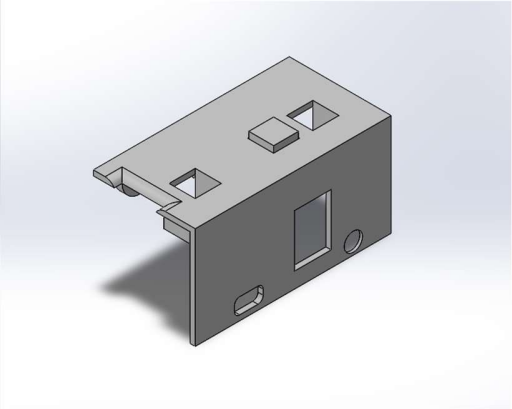


Figure 20: base back cover

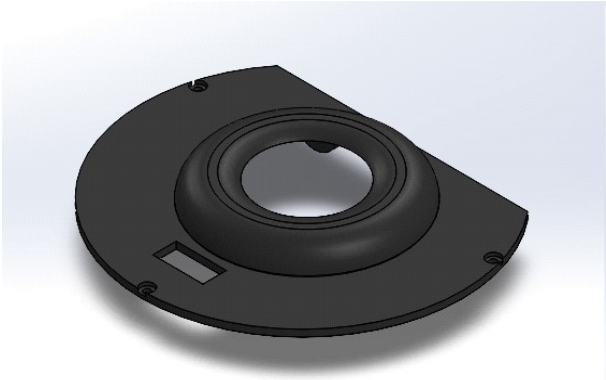


Figure 21: base top cover

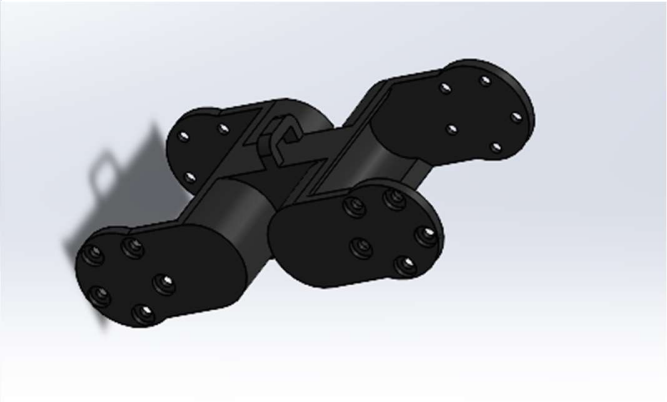


Figure 22: arm

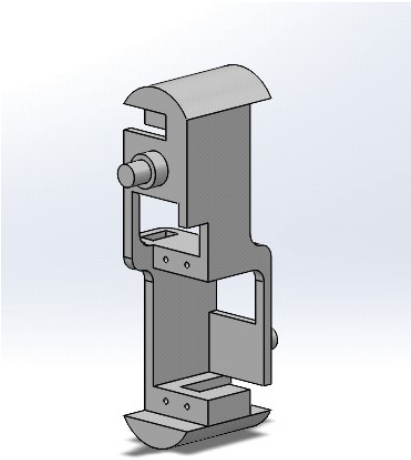


Figure 23: arm head joiner

Code for transmitter:

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

#define changeRate 0.05

esp_now_peer_info_t receiver;
uint8_t esp32receiver[] = {0xB0, 0xA7, 0x32, 0xF3, 0x64, 0x31};

int potPins[] = {36, 39, 34, 35, 32, 33};

int potMin[] = {1250,0,1900,4095,1460,4095};
int potMax[] ={4095,2460,0,1440,4095,3280};

// LED rgb pin def
const int redwire = 15;
const int greenwire = 2;
const int bluewire = 4;

uint8_t data[6];

void writeLed(int, int, int);

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  // char macStr[18];
  // snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac_addr[0],
mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  // Serial.print("Last Packet Sent to: "); Serial.println(macStr);
  // Serial.print("Last Packet Send Status: "); Serial.println(status == ESP_NOW_SEND_SUCCESS
? "Delivery Success" : "Delivery Fail");
}

void setup(){

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  esp_now_init();
  memcpy(receiver.peer_addr, esp32receiver, 6);
  receiver.channel = 1;
  receiver.encrypt = false;
  esp_now_add_peer(&receiver);
  esp_now_register_send_cb(OnDataSent);

  for(int i=0; i<6; i++){
    pinMode(i, INPUT);
  }
  Serial.begin(115200);

  writeLed(0, 0, 255);
```

```
}

int pre;
int current;

void loop(){
  for(int i=0; i<6; i++){
    //int temp = map(analogRead(potPins[i]), potMin[i], potMax[i], 0, 180);

    //Serial.print("temp = "); Serial.print(temp);
    //data[i] = temp;
    //Serial.print("t=");Serial.print()

    pre = data[i];
    int temp = map((i == 0 || i == 1 || i == 4)?(analogRead(potPins[i]) < potMin[i] ?
potMin[i] : analogRead(potPins[i])):analogRead(potPins[i]), potMin[i], potMax[i], 0, 200);
//0 1 4

    //(i == 0 || i == 1 || i == 4)?(analogRead(potPins[i]) < potMin[i] ? potMin[i] :
analogRead(potPins[i])):analogRead(potPin[i])


    current = (temp * changeRate) + (pre * (1-changeRate));

    pre = current;

    data[i] = current;
    if(data[i]>180){
      data[i] = 180;
    } else if(data[i]<0){
      data[i] = 0;
    }

    // if(i==5){
    //   data[i] = map(data[i], 0, 180, 0, 102);
    // }

    // if(abs(data[i]-temp) > 2){
    //   data[i] = temp;
    // } else if(temp == 0) {
    //   data[i] = 0;
    // } else if(temp == 180){
    //   data[i] = 180;
    // }


    //Serial.print(" Data = ");
    Serial.print(data[i]); Serial.print(", ");
  }
```

```
  Serial.println();
  esp_now_send(esp32receiver, data, sizeof(data));
  //delay(200);
}

void writeLed(int r, int g, int b){
  analogWrite(redwire, r);
  analogWrite(greenwire, g);
  analogWrite(bluewire, b);
}
```

Code for receiver:

```
#include <ESP32Servo.h>
#include <esp_now.h>
#include <WiFi.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"

//#include <Ramp.h>

#include <ESPAsyncWebServer.h>
#include <ArduinoWebsockets.h>
#include <ArduinoJson.h>

const char *ssid = "Finance_Mantra_2.4";
const char *password = "Whatdahellisthes";

esp_now_peer_info_t peerInfo;
uint8_t broadcastAddress[] = {0xCA, 0xC9, 0xA3, 0x60, 0x2B, 0xA5};
uint8_t data[] = {0};

AsyncWebServer server(80);
AsyncWebSocket myWebSocket("/ws");

bool clientConnected = false;


// channel to look data for
#define CHANNEL 1


// Servo def
Servo servo[6];
int servopins[] = {25, 26, 27, 14, 12, 13};

int servopotpins[] = {36, 39, 34, 35, 32, 33};

int minVal[] = {190, 149, 205, 171, 171, 336};
int maxVal[] = {3530, 3507, 3514, 3376, 3376, 2396};

int servoAttached = 0;


// Push button pin def
const int pushbttn1 = 21;
const int pushbttn2 = 22;


// LED rgb pin def
```

```
const int redwire = 15;
const int greenwire = 2;
const int bluewire = 4;

char path[] = "/pos.txt";

// Functions Dec
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status);
void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len);
void readServoPosition(void);
void writeLed(int r, int g, int b);
void removeTrailAtEOF(void);
void onWebSocketEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType
type, void *arg, uint8_t *data, size_t len);
//void recordMove(void);

void setup() {
  Serial.begin(115200);

  WiFi.mode(WIFI_AP);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");


  esp_now_init();

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 1;
  peerInfo.encrypt = false;

  esp_now_register_recv_cb(OnDataRecv);
  esp_now_register_send_cb(OnDataSent);

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

  for(int i=0; i<6; i++){
    servo[i].attach(servopins[i]);
    // digitalWrite(servopotpins[i], HIGH);
    pinMode(servopotpins[i], INPUT);
  }
```

```
servoAttached = 1;

pinMode(pushbttn1, INPUT);
pinMode(pushbttn2, INPUT);

if(!SD.begin()){
  Serial.println("SD card initialization failed!");
  writeLed(255, 0, 0);
  while(1){
    //
  }
}

 // Serve HTML page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
  request->send(200, "text/html", R"rawliteral(
    <html>
      <head>
        <script>
          var socket = new WebSocket('ws://' + window.location.hostname + '/ws');

          function updateServos() {
            var servoPos = [
              document.getElementById('servo1').value,
              document.getElementById('servo2').value,
              document.getElementById('servo3').value,
              document.getElementById('servo4').value,
              document.getElementById('servo5').value,
              document.getElementById('servo6').value
            ];

            var data = {
              servo1: parseInt(servoPos[0]),
              servo2: parseInt(servoPos[1]),
              servo3: parseInt(servoPos[2]),
              servo4: parseInt(servoPos[3]),
              servo5: parseInt(servoPos[4]),
              servo6: parseInt(servoPos[5])
            };

            socket.send(JSON.stringify(data));
          }
        </script>
      </head>
      <body>
        <h1>Servo Control</h1>

        <label for='servo1'>Servo 1</label>
```

```
          <input type='range' id='servo1' min='0' max='180' value='90'
oninput='updateServos()'><br><br>

          <label for='servo2'>Servo 2</label>
          <input type='range' id='servo2' min='0' max='180' value='45'
oninput='updateServos()'><br><br>

          <label for='servo3'>Servo 3</label>
          <input type='range' id='servo3' min='0' max='180' value='0'
oninput='updateServos()'><br><br>

          <label for='servo4'>Servo 4</label>
          <input type='range' id='servo4' min='0' max='180' value='40'
oninput='updateServos()'><br><br>

          <label for='servo5'>Servo 5</label>
          <input type='range' id='servo5' min='0' max='180' value='90'
oninput='updateServos()'><br><br>

          <label for='servo6'>Servo 6</label>
          <input type='range' id='servo6' min='24' max='102' value='24'
oninput='updateServos()'>
        </body>
      </html>
    )rawliteral");
  });

  // Start WebSocket server
  myWebSocket.onEvent(onWebSocketEvent);
  server.addHandler(&myWebSocket);

  // Start the server
  server.begin();

  writeLed(0, 0, 255);
}

int deleted = 0;

void loop() {
  if(digitalRead(pushbttn1) && !digitalRead(pushbttn2)){
    if(deleted == 0){
      SD.remove(path);
      deleted = 1;
    }
    for(int i=0; i<6; i++){
      servo[i].detach();
    }
    servoAttached = 0;
```

```
    writeLed(255, 0, 0);
    readServoPosition();
    //removeTrailAtEOF();
  }

  if(!digitalRead(pushbttn1) && digitalRead(pushbttn2)){
    removeTrailAtEOF();
    if(!servoAttached){
      for(int i=0; i<6; i++){
        servo[i].attach(servopins[i]);
      }
    }
    servoAttached = 1;
    writeLed(0, 255, 0);
    readFile(SD, path);
    deleted = 0;
  }
  //esp_now_send(broadcastAddress, data, sizeof(data));
  // Serial.print(analogRead(36));
  // Serial.print(", ");
  // Serial.print(analogRead(39));
  // Serial.println();
  // delay(50);
}


// Functions def

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");
}

void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  if(!servoAttached){
    for(int i=0; i<6; i++){
        servo[i].attach(servopins[i]);
    }
    servoAttached = 1;
  }
  for (int i = 0; i < 6; i++) {
    Serial.print(data[i]); Serial.print(", ");
    if(i==5){
      int temp = map(data[i], 0, 180, 24, 102);
      servo[i].write(temp);
    } else{
      servo[i].write(data[i]);
    }
    //servo[i].write(data[i]);
```

```
  }
  Serial.println();
}

void readServoPosition(void){

    String dataString = "";
    for(int i=0; i<6; i++){
      int temp = analogRead(servopotpins[i]);
      int maptemp = map(temp, minVal[i], maxVal[i], 0, 180);
      dataString += String(maptemp);
      dataString += ",";
      // Serial.print(maptemp);
      // Serial.print(", ");
    }
    //writeFile(SD, "/pos.txt", String(temp).c_str());
    dataString += "\n";
    writeFile(SD, path, dataString);
    // Serial.println();

  writeLed(0, 0, 255);
}

void writeLed(int r, int g, int b){
  analogWrite(redwire, r);
  analogWrite(greenwire, g);
  analogWrite(bluewire, b);
}

void writeFile(fs::FS &fs, const char *path, const String &message) {
  //Serial.printf("Writing file: %s\n", path);
  //writeLed(255, 0, 0);
  File file = fs.open(path, FILE_APPEND);
  if (!file) {
    Serial.print("Failed to open file for writing");
    return;
  }
  if (file.print(message.c_str())) {
    //Serial.print("File written");
  } else {
    Serial.print("Write failed");
  }
  file.close();
  //delay(50);
}

void readFile(fs::FS &fs, const char * path){
  //writeLed(0, 255, 0);
  //Serial.printf("Reading file: %s\n", path);
```

```
  // Serial.println();
  // Serial.println("File Started: ");
  File file = fs.open(path);
  if(!file){
    Serial.println("Failed to open file for reading");
    return;
  }
  int i = 0;
  //Serial.print("Read from file: ");
  while(file.available()){
    //Serial.write(file.read());
    String buffer = file.readStringUntil(',');
    int rot = buffer.toInt();
    // int rot = (buffer.toInt(), 3450, 190, 0, 180);
    Serial.print(rot);
    // Serial.print("-");
    // Serial.print(i);
    servo[i].write(rot);
    delay(1);
    Serial.print(", ");
    if(i<5){
      i++;
    }
    else{
      i = 0;
      Serial.println();
    }
  }
  file.close();
  writeLed(0, 0, 255);
}

void removeTrailAtEOF() {
  File file = SD.open(path, FILE_READ);

  if (!file) {
    Serial.println("Failed to open file for reading");
    return;
  }

  int fileSize = file.size();
  if (fileSize > 0) {
       char* fileContent = new char[fileSize + 1];

    file.readBytes(fileContent, fileSize);
    fileContent[fileSize] = '\0';
    file.close();
    if (fileSize >= 2 && fileContent[fileSize - 2] == ',') {
      fileContent[fileSize - 2] = '\0';  // Remove the trailing comma
```

```
    }
    File outFile = SD.open(path, FILE_WRITE);

    if (outFile) {      outFile.print(fileContent);

      // Close the output file
      outFile.close();

      //Serial.println("Trailing comma removed successfully.");
    } else {
      //Serial.println("Failed to open output file for writing.");
    }

    // Deallocate the buffer
    delete[] fileContent;
  } else {
    //Serial.println("File is empty.");
  }
}

void onWebSocketEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType
type, void *arg, uint8_t *data, size_t len) {
  if (type == WS_EVT_CONNECT) {
    Serial.println("WebSocket client connected");
    clientConnected = true;
  } else if (type == WS_EVT_DISCONNECT) {
    Serial.println("WebSocket client disconnected");
    clientConnected = false;
  } else if (type == WS_EVT_DATA && clientConnected) {
    // Parse JSON data from the WebSocket message
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, (const char *)data);
    servo[0].write(doc["servo1"]);
    servo[1].write(doc["servo2"]);
    servo[2].write(doc["servo3"]);
    servo[3].write(doc["servo4"]);
    servo[4].write(doc["servo5"]);
    servo[5].write(doc["servo6"]);
  }
}
```

Schematic Diagram:

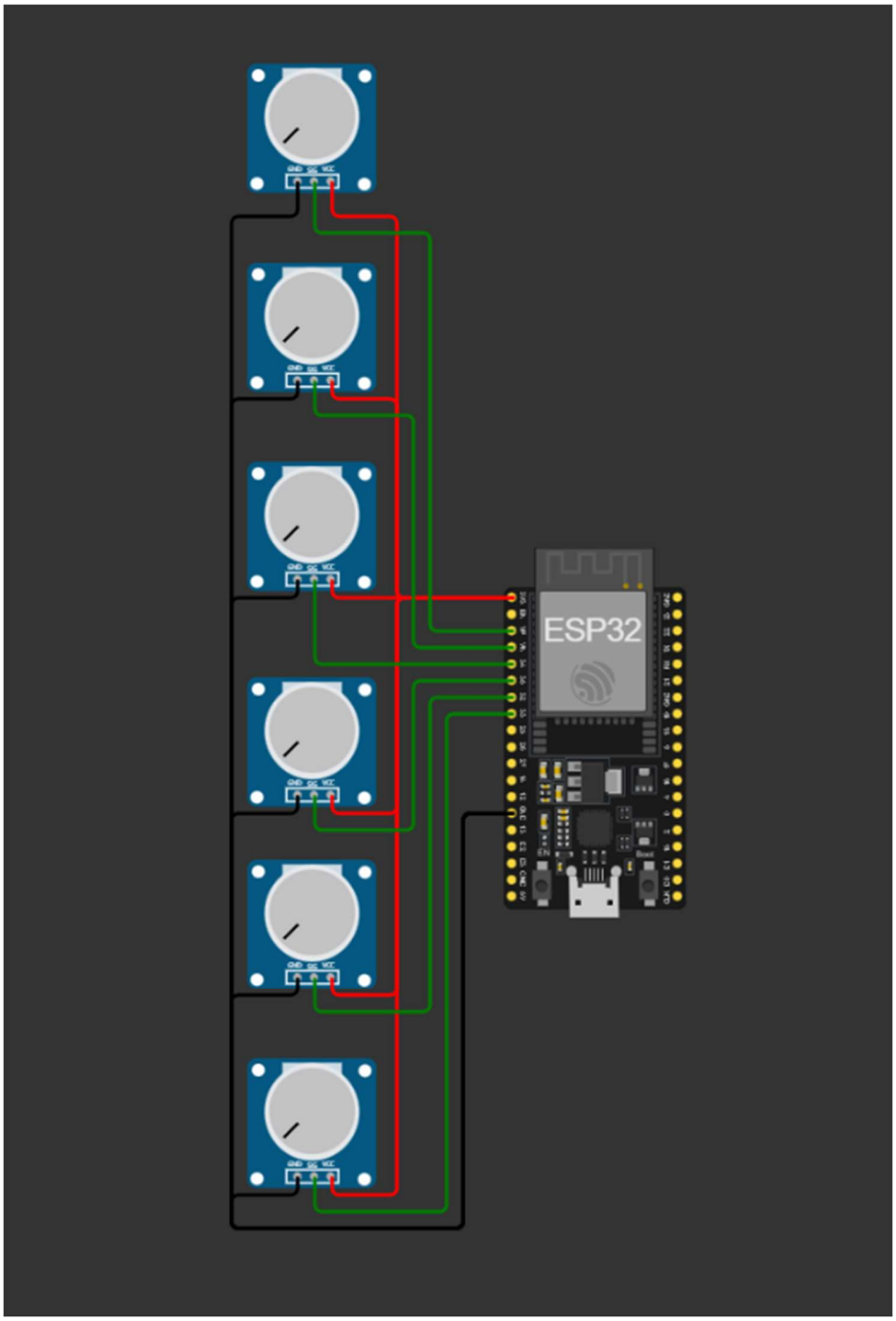


Figure 24: schematic for transmitter

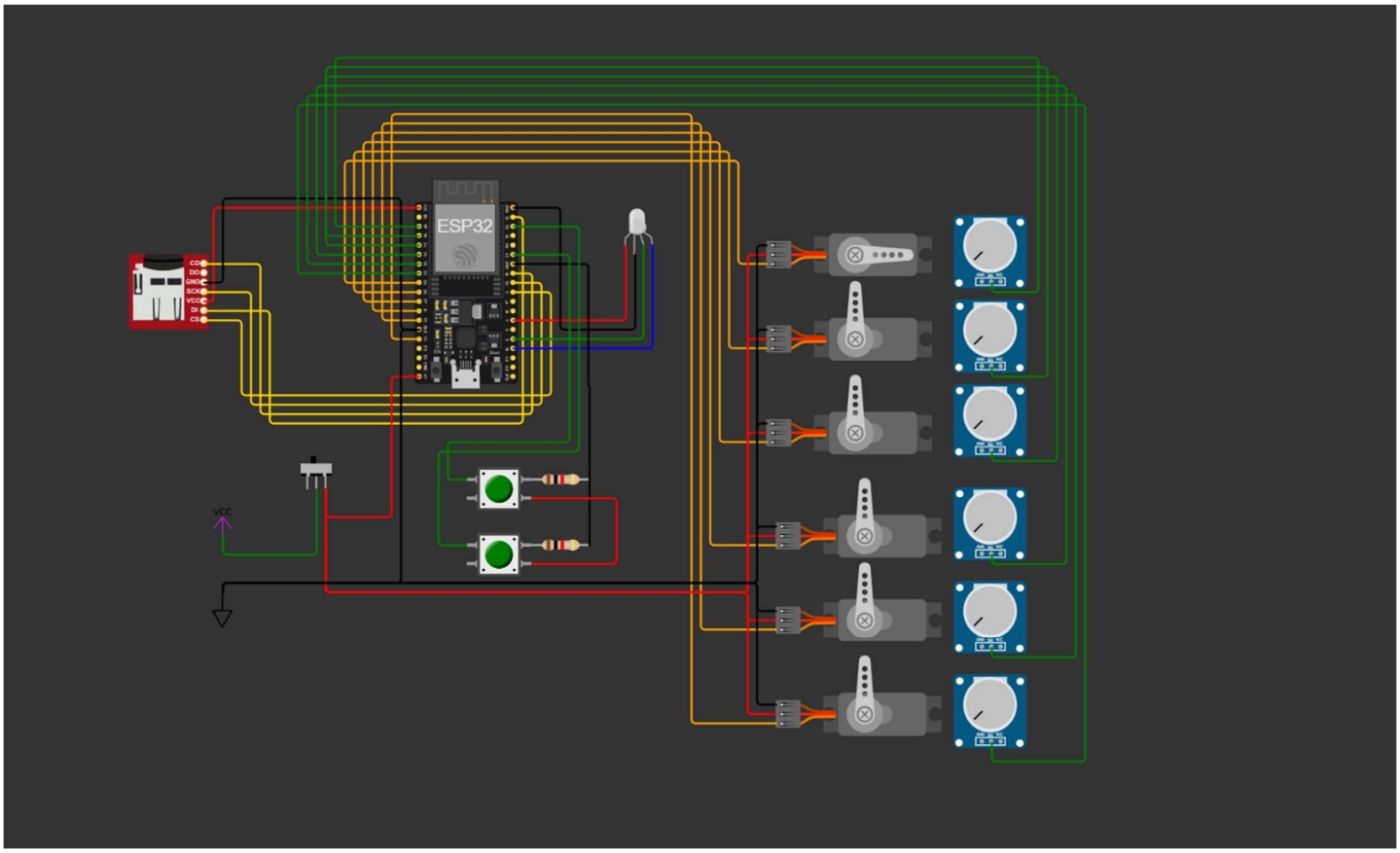


Figure 25: schematic for receiver

Components Used:

-Receiver

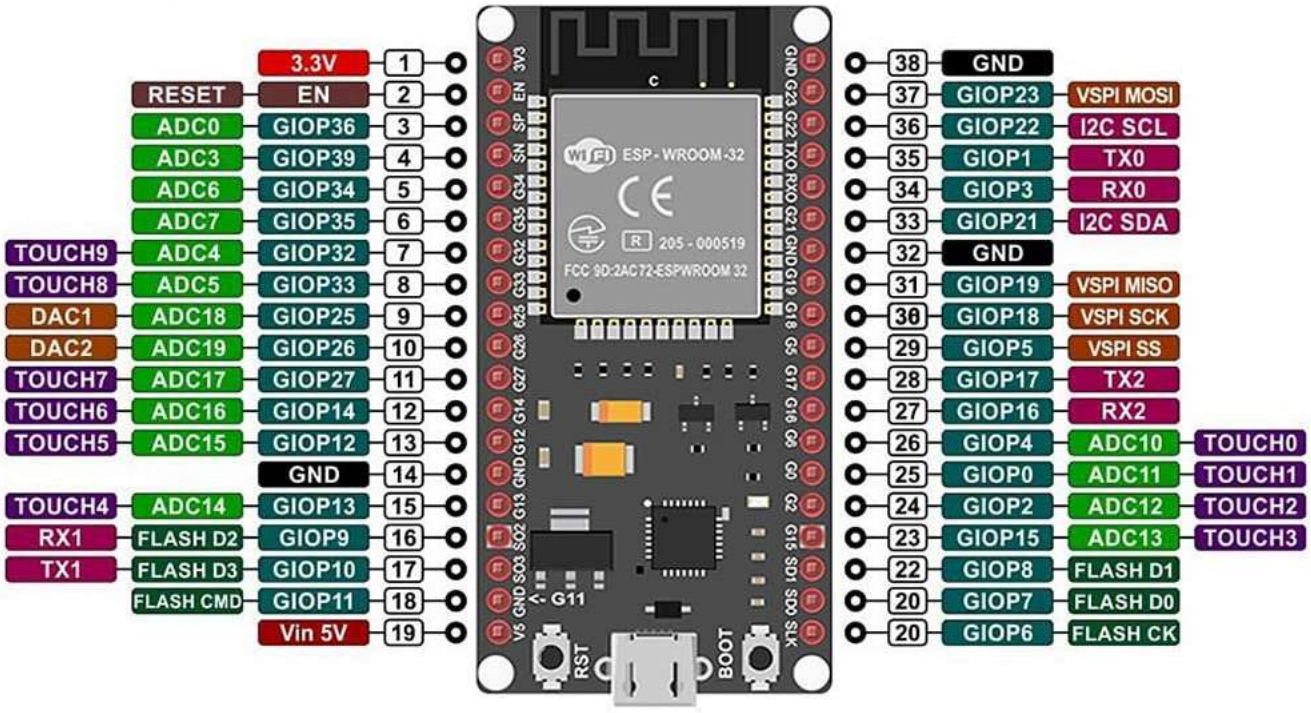
- ESP-32 Dev Kit v1 (Microcontroller)
- MG996R Servo Motor
- MG90S Micro Servo Motor
- RGB Led
- Reset Switches
- Micro-SD card Module

-Transmitter

- ESP-32 Dev Kit v1 (Microcontroller)
- Trimpot Potentiometer (5KΩ)
- Charging Module
- 18650 Li-ion Battery

ESP32 Dev Kit v1

ESP32 is one of the development board created using ESP-WROOM-32 Module. It is based on ESP32 microcontroller that has the functionality of WiFi, Bluetooth, Low power Support on a single chip.



Specification:

- Operating Voltage: 3.3V
- Input Voltage: 7-12v
- Digital I/O Pins (DIO): 25
- Analog Input Pins (ADC): 6 (12-bit)
- Analog Output Pins (DAC): 2
- UARTs: 3
- SPIs: 2
- I2C: 3
- Flash Memory: 4 MB
- Clock Speed: 240 Mhz

MG996R Servo Motor

MG996R is all metal gear servo motor which is commonly used in DIY projects like in this one.

Specification:

- Operating Voltage: 5V
- Current: 2.5A (6V)
- Stall Torque: 9.4 kg/cm (at 4.8V)
- Maximum Stall Torque: 11 kg/cm (6V)
- Operating speed is 0.17 s/60°
- Gear Type: Metal
- Rotation : 0°-180°
- Weight of motor : 55gm

MG90S Micro Servo Motor

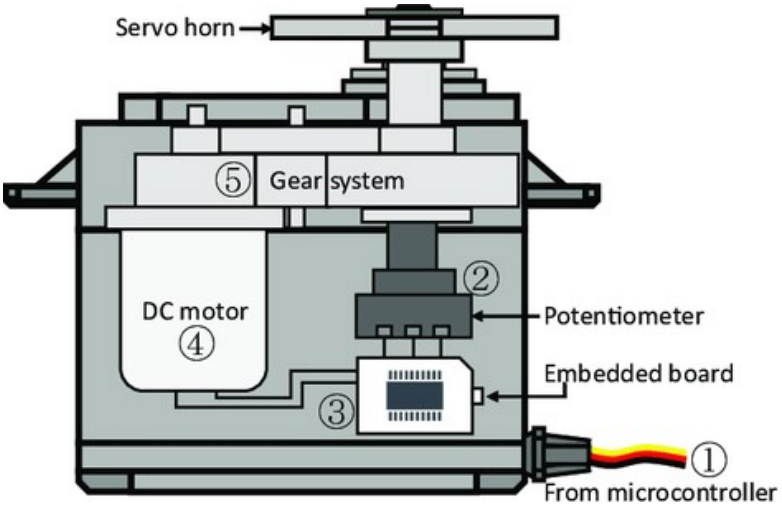
MG90S is yet another servo motor with lesser size and lower torque.

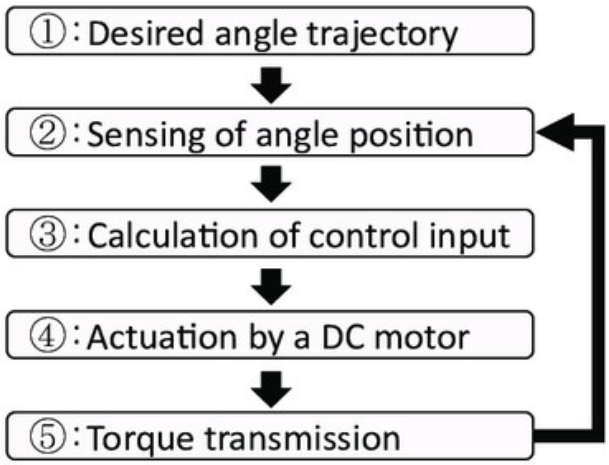
Specification:

- Operating Voltage: 5V
- Current: 2.5A (6V)
- Stall Torque: 1.8 kg/cm (at 4.8V)
- Maximum Stall Torque: 2.2 kg/cm (6V)
- Operating speed is 0.10 s/60°
- Gear Type: Metal
- Rotation : 0°-180°
- Weight of motor : 13.4gm

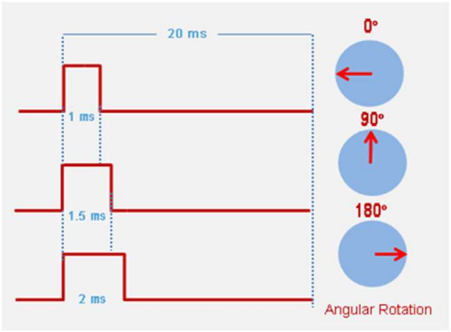
Working of Servo Motor:

The shaft of a servo motor is directly connected to a potentiometer. As the shaft rotates, the resistance of potentiometer change, so the circuit precisely detect the rotation making it able to make movements in certain direction.





Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. A servo motor can usually only turn 90° in either direction for a total of 180° movement. The motor's neutral position is defined as the position where the servo has the same amount of potential rotation in the both the clockwise or counter-clockwise direction. The PWM sent to the motor determines position of the shaft, and based on the duration of the pulse sent via the control wire; the rotor will turn to the desired position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90° position. Shorter than 1.5ms moves it in the counter clockwise direction toward the 0° position, and any longer than 1.5ms will turn the servo in a clockwise direction toward the 180° position.

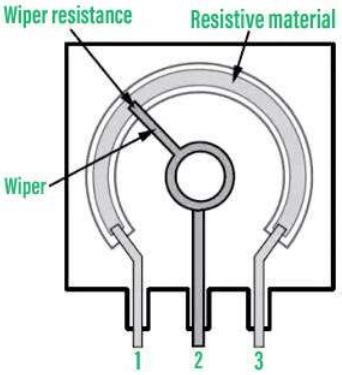


Working of Trimpot Potentiometer:

A trimpot potentiometer, also known as a trimmer potentiometer or trim pot, is a type of variable resistor that allows for manual adjustment of its resistance value. It is commonly used for calibration, tuning, and adjustment in electronic circuits.

Physical Structure:

Trimpot potentiometers typically consist of a resistive element that can be adjusted by rotating a small knob or screw. The resistive element is often made of a resistive material such as carbon, and the wiper (a conductive material) moves along the resistive track.



Three Terminal Configuration:

Trimpot potentiometers typically have three terminals – two outer terminals and one middle terminal (wiper). The two outer terminals are connected to the resistive element, while the middle terminal (wiper) is connected to the moving part.

Variable Resistance:

The resistance between one of the outer terminals and the wiper terminal changes as the potentiometer is adjusted. By rotating the knob or screw, the wiper moves along the resistive track, changing the portion of the resistive element in the circuit. This movement alters the resistance between the wiper and the fixed terminal, providing a variable resistance.

Working of controller arm:

Whole working process takes place in 5 actual steps:

Step-01: (reading raw value)

Transmitter ESP reads **raw value of potentiometer.** Based on its rotation. Potentiometer creates variable resistance for each rotation position. Variable resistance creates variable voltage value across it signal and GND pins. This voltage is detected by the ESP-32 as an analog value but microcontroller does not seem to understand those analog value. So the 12 bit ADC converter used in ESP-32 converts this analog value. Since 2^12 = 4096. The digital value ranges from 0~4095.

Step-02: (mapping the raw value)

One the raw value raw value is read, **mapping processes** starts. Here the value ranging from 0~4095 is mapped to value ranging from 0~180. map() function is a versatile tool that allows to scale or map a range of values from one set to another. This can be understood as a speedometer analogy.

Step-03: (filtering sharp changes)

Noise/Jitter filtering algorithm.

```
float decayRate = 0.05;
int prev_value;
int current_value;
// On loop, smooth the movements toward the target
current_value = (target_angle * decayRate) + (prev_value * (1-decayRate)
prev_value = current_value
```

Here, the noise from the potentiometer is filtered by making only small increments to the last value, in this case 5%.

Figure 26: Before

Figure 27: After

Step-04: (transmitting the read data)

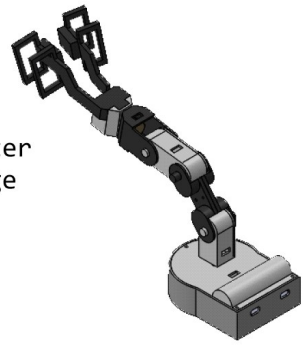
The above mentioned 3 steps repeats 6 times for each potentiometer each instance. Creating an array of data of length 6 and transmits it to the receiver paired ESP32.

For example:

`int data[6] = {45,35,70,120,45,102};` is sent to the receiver pair.

Working of controller arm visualised:

takes input from potentiometer as analog value (i.e. voltage ranging from 0~5V



Internal Processing:

1. Map the value read ie. 0-4095 to 0-180
2. Filter noise using aforementioned noise filtering algorithm.
3. Transmits the filtered data to its pair

Working of main arm:

First Mode (Being Controlled using controller arm):

Step-01: (Listening for any incoming data)

Creates a listening channel for receiving all the incoming data.

Attaches all the motor to its respective pins.

Step-02: (rotating the servo)

When any data is received from the controller arm, For example, previously sent int data[6] = {45,35,70,120,45,102};

It goes through a loop rotating each servo to respective position.

Second Mode (Learning and repeating movements):

Step-01: (Reading the potentiometer value of servo)

Reading the value of potentiometer (i.e. 0~4095). Mapping it to values from 0~180 same as in the case of controller arm when record button is pressed.

Step-02: (Writing it to the connected SD-card)

Opening a file named “pos.txt” on the connected SD-card in writing mode. **Write the already mapped value** separating it by comma (,) and adding a new line after each reading session. For Example:

1,127,13,12,66,102,

1,126,14,13,66,102,

1,127,13,13,66,102,

1,127,13,12,66,102,

1,127,12,12,66,102,

1,127,12,12,66,103,

1,127,12,12,66,102,

Step-03: (Replaying the recorded motion)

When the recording button is pressed, previously created “pos.txt” is opened in reading mode. **Reading each value and rotating the servos** to the read position. This process continues till the EOF and loops infinitely until microcontroller is reset.

Note:

Few steps are not mentioned to reduce the complexity. Like removing the trailing comma at the EOF to prevent the servo motor from showing un-natural/un-defined motion.

Third Mode (Controlled through Wi-Fi):

Step-01: (Setting up connection)

Connects to pre-determined Wi-Fi network. And broadcasts a web address with its port no. 80 open.

Step-02: (Listening for incoming data)

When a client makes request to the web address, this mode is activated. Deploying a websocket on the client browser. Allowing real time communication.

Step-03: (Reading value from client)

When the slider on the front end is moved, using JavaScript the value (0~180) is read and sent to the microcontroller through the websocket.

Step-04: (Rotating the servo)

After reading the value, rotating the servo to the read value (0~180).