Universidad
Rey Juan Carlos

C.O.L.O.R.E.S

GRUPO 13
LAURA REYES DEL RÍO
ADAY RODADO COELLO
GUSTAVO MARGARIDA GÓMEZ

ÍNDICE

1. **INTRODUCCIÓN Y OBJETIVOS** 2
2. **IMPLEMENTACIÓN** 2
3. **COSTES DE MATERIALES** 3
4. **PASOS DADOS** 4
5. **REPARTO DE TAREAS** 8
6. **PROBLEMAS Y SUS SOLUCIONES** 8
7. **CÓDIGO** 10
   a. CÓDIGO ARDUINO 10
   b. CÓDIGO EN PYTHON 22
8. **CASOS DE USO** 29
9. **POSIBLES MEJORAS FUTURAS** 30

1. INTRODUCCIÓN Y OBJETIVOS

Nuestro proyecto se centra en el uso de una aplicación con detección de colores con la cual realizar distintas actividades, entre ellas:

- **Sistema de manipulación remoto:** Una mano que reacciona según el movimiento de colores realizado en pantalla.
- **RGB reactivo**: Led RGB que cambia de color en función de los colores en pantalla.
- **Réplica de un potenciómetro:** Zumbador que cambia de volumen y tonalidad en función del lugar del color en pantalla.

Con estas actividades, nuestro objetivo principal es poder realizar acciones de la vida cotidiana como subir el volumen de un altavoz, cambiar de color las distintas tiras o lámparas LED de una habitación o el simple gesto de recoger y dejar objetos con las manos, de forma remota.

En adición, un objetivo a tener en cuenta es el aprendizaje del uso de librerías de OpenCV.

2. IMPLEMENTACIÓN

Como parte de los componentes necesarios para todo el proyecto encontramos: la placa Arduino UNO, la breadboard, cables y resistencias. Además de lo nombrado anteriormente, en el resto de los casos, encontramos diferentes materiales.

Primero, para poder diferenciar los 3 modos, serán necesarios 3 leds para identificar visualmente en qué modo estamos y un botón para poder cambiar entre modos.

Para nuestro primer modo, encontramos 3 servomotores conectados a la breadboard gracias a sus 3 cables conectores siendo el negro el dirigido a la toma de tierra, el rojo a la toma de corriente y el amarillo dirigido a cada uno de sus pines correspondientes en la placa. La energía suministrada por la placa no es suficiente para hacer funcionar correctamente todos los servomotores, por lo que, gracias a la protoboard MB102 y una fuente de alimentación externa conectada a la misma, podemos realizar el movimiento de los servomotores perfectamente. Para el material de montaje optamos por realizar la mano con cartón, además de conectar cada servo con cuerdas a los distintos dedos. Si bien, para poder realizar el movimiento de los dedos correctamente, se añadieron piedras en las puntas de los dedos a modo de contrapeso.

Para el segundo y tercer modo, optamos por coger una minibreadboard para mantener un orden visual. Por tanto, en el segundo modo, tan solo necesitábamos un buzzer pasivo conectado a su respectivo pin en la placa de Arduino y toma de tierra.

Finalmente, en el tercer modo tan solo necesitábamos el led RGB conectando cada uno de sus leds de color internos (rojo, azul y verde) a sus respectivos pines en la placa de Arduino y toma de tierra.

3. COSTES DE MATERIALES

Gracias a esta tabla observamos lo que nos ha costado a nosotros el proyecto siendo en total 43€ debido a que aquellos productos de precio gratuito para nosotros son aquellos materiales proporcionados por la universidad. Si quisiésemos realizar nosotros este proyecto sin dichos materiales, el precio oscilaría los 55-75€ dependiendo del lugar de obtención de materiales y de la calidad de los mismos.

MATERIAL	CANTIDAD	PRECIO
Servomotores	4-5	15-20
Cables	Aprox. 30	Gratis
Fuente de alimentación	1	20
MB102	1	Gratis
Breadboard	1	Gratis
MiniBreadboard	1	Gratis
Placa Arduino UNO	1	Gratis
Botón	1	Gratis
Leds y Led RGB	4	Gratis

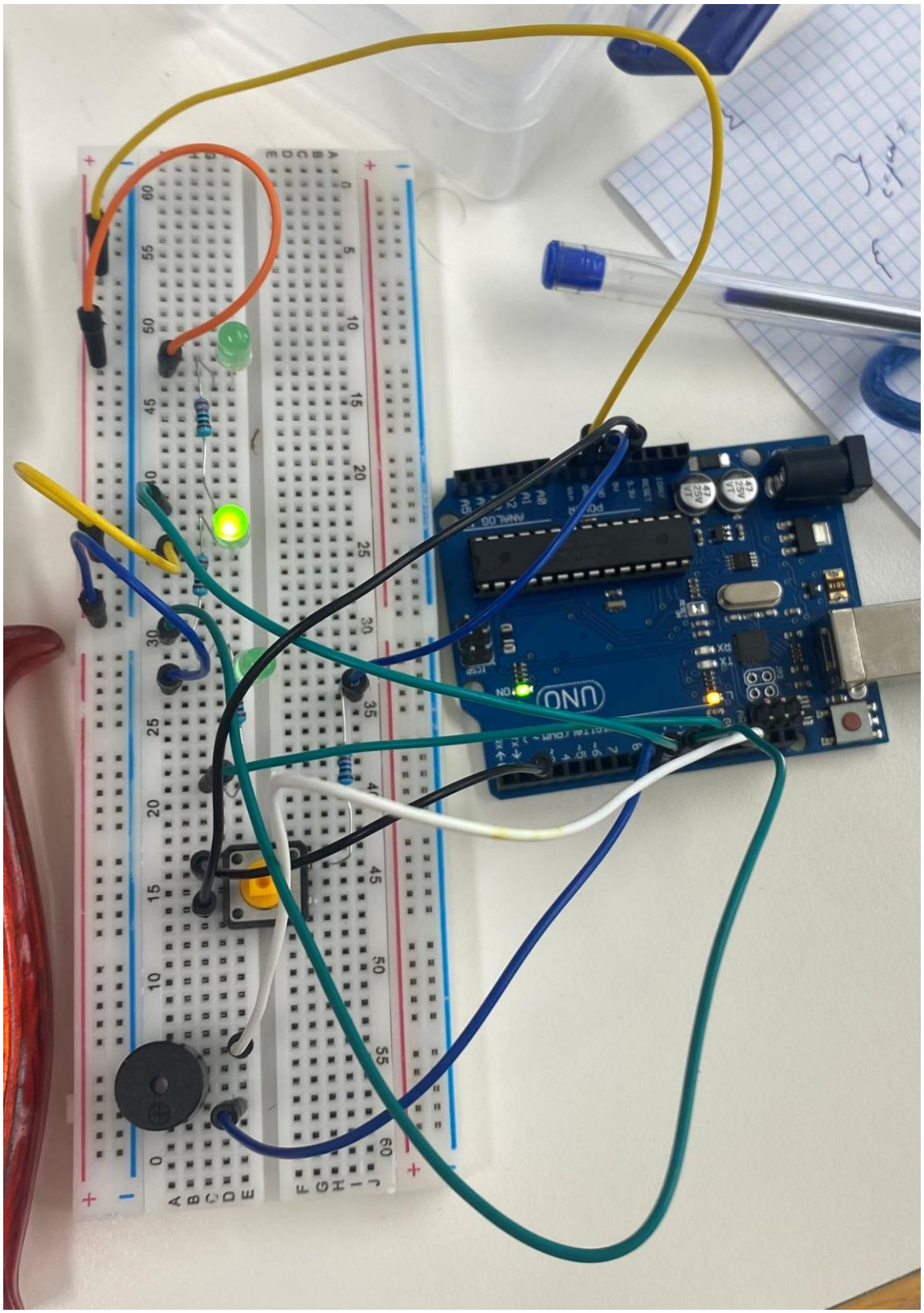
Zumbador	1	Gratis
Resistencias	7	Gratis
Materiales de construcción	Necesarios	3

4. PASOS DADOS

Primero, comenzamos con una lluvia de ideas entre las que encontramos las dos opciones más llamativas como un laberinto con acertijos a resolverlo a través de la cámara con las distintas opciones y colores o, la opción finalmente escogida, siendo así la elección entre 3 modos: réplica de una mano, réplica de un potenciómetro y led RGB reactivo.

Para ello, primero empezamos programando la parte de detección de colores por la cámara. Para comprobar que los detectaba correctamente y funcionaba como esperábamos, decidimos programar el modo número 2 de la réplica del potenciómetro. Una vez comprobamos que podíamos realizarlo de forma sencilla, procedimos con los modos.

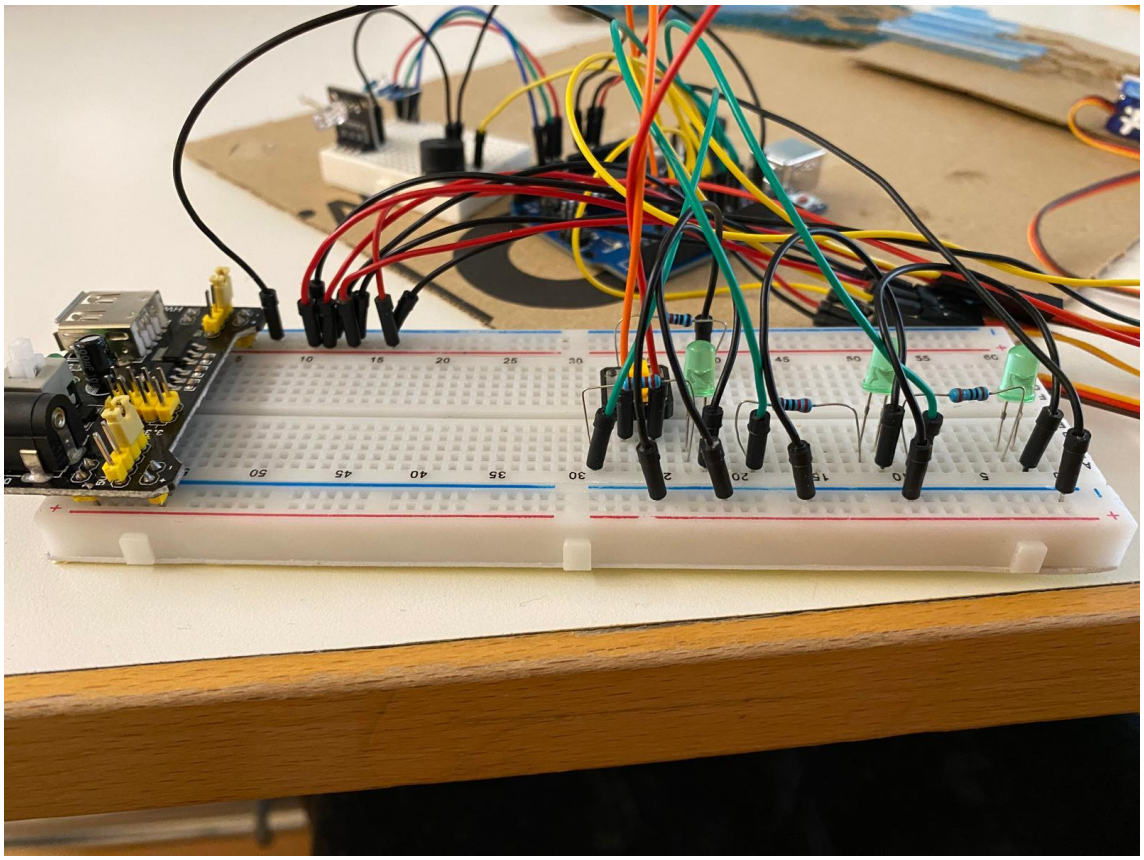
Para poder diferenciar los modos tenemos 3 leds y un botón por lo que primero programamos la existencia de los modos y encender el led cambiando de modo a través del botón añadiendo lo programado anteriormente para el modo número 2 en el condicional cuando encendemos el segundo led. Por lo que el circuito obtenido era este:

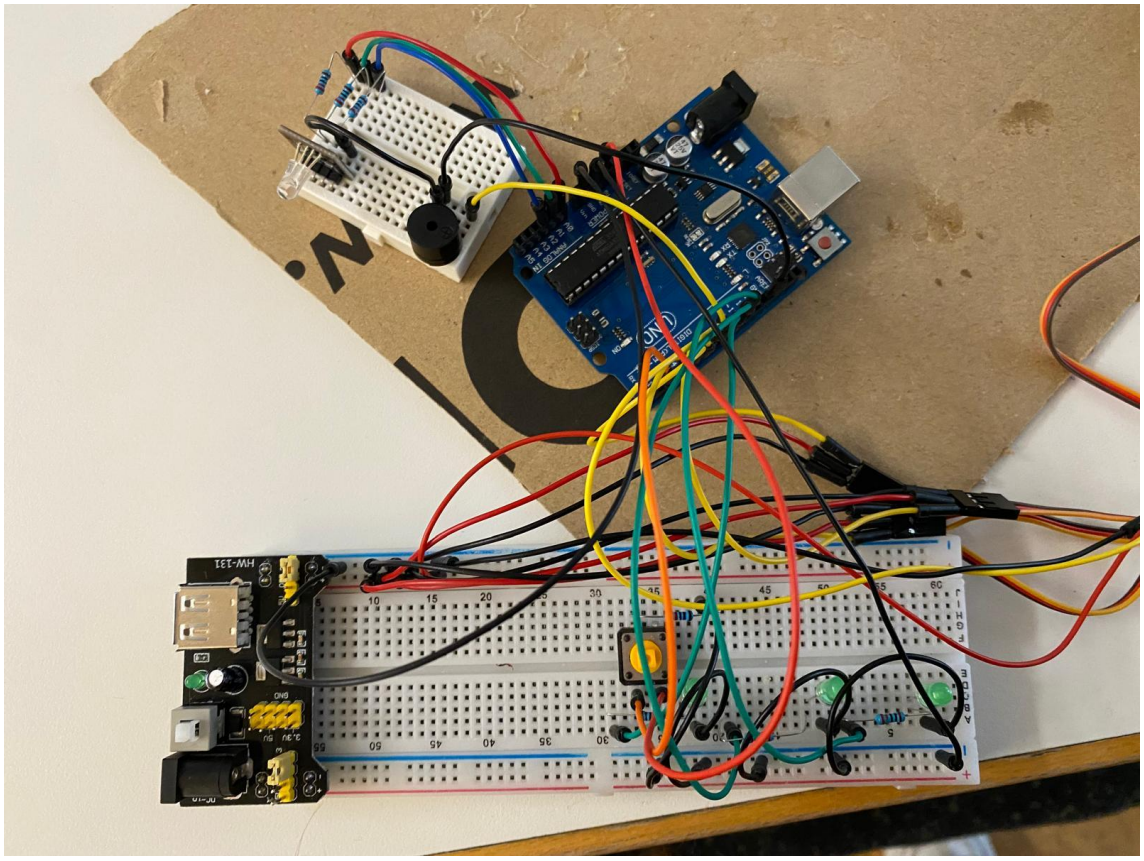


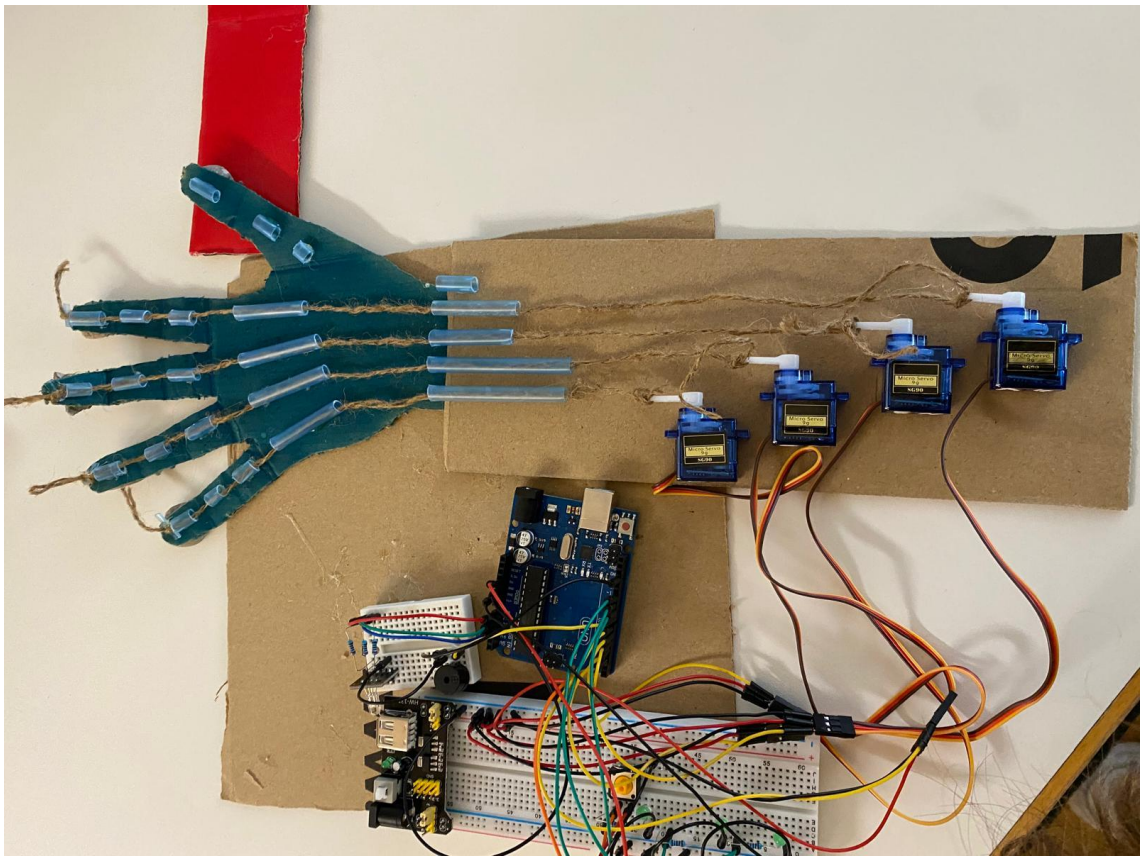
Más tarde, procedimos a la programación de los servomotores para que estos se moviesen en 180 grados y se moviesen según los datos recibidos por el programa en Python. Una vez los servomotores funcionaban correctamente proseguimos con la creación de la mano:

Una vez teníamos la mano construida, teníamos que comprobar el lugar de cada servomotor para que estos no se entorpeciesen entre sí, además de la correcta calibración de los mismos. Con todo listo, nos faltaba montarlo.

En adición, procedimos a añadir la tercera opción del led RGB y reorganizar todos los cables para una mejor visualización del circuito. Con el resultado final siendo:




HW-131



5. REPARTO DE TAREAS

En esta práctica el reparto de tareas ha sido equitativo debido a que todos los miembros del grupo nos hemos organizado en reuniones presenciales para poder realizar el proyecto de forma correcta y al unísono. Esto nos proporcionó a todos los miembros la posibilidad de aprender a realizar tanto el código como las conexiones físicas del circuito de nuestro proyecto.

6. PROBLEMAS Y SUS SOLUCIONES

Nuestros problemas podemos dividirlos en dos categorías: físicos y de código.

En cuanto a los físicos se refiere nos encontramos con:

- Pines colocados en posiciones erróneas por despiste.
- Problemas relacionados con la distribución y suministro de alimentación. Al principio, no estábamos seguros de si la placa iba a poder enviar la suficiente corriente para los servomotores y a la hora de la práctica nos lo confirmó, por lo que optamos por utilizar la MB102 otorgada por la universidad y comprar una fuente de alimentación para conseguir que todos los servomotores se moviesen correctamente.

- Servomotor quemado por cortocircuito. Debido a una mala conexión del servomotor este se nos quemó durante el proceso.
- Problemas al comprar servomotores de rotación continuos y no posicionales. En un principio, quisimos comprar servomotores posicionales con los que fuese más sencillo programar el movimiento de los dedos. Sin embargo, al comprar erróneamente los continuos, optamos por calibrarlos manualmente controlando la velocidad y la duración a la que se movían cada uno de los servomotores.
- Problemas al regular el movimiento de la mano por el peso de los dedos. Tuvimos problemas tanto con el cierre como con la regresión natural de la mano. Esto se debía a que, sin las piedras en los dedos, cuando se intentaba regresar a la forma natural de la mano, no regresaba completamente ya que la cuerda no tenía la fuerza suficiente. Optamos por utilizar piedras pequeñas en la punta de los dedos a modo de contrapeso. Esto ocasionó, que, a la hora de cerrar, debiésemos tener cuidado con la calibración del cierre ya que, si tenía mucho peso, la mano era incapaz de volver a su posición natural. Por tanto, tuvimos que recalibrar todos los servomotores con distintos tiempos y velocidades a la hora del cierre como a la de regresión.

En cuanto a los de código se refiere nos encontramos con:

- Diversos problemas con bucles.
- Controlar la gama de colores correctamente. Debido a los cambios de luminosidad en diferentes sitios de pruebas, ciertos colores se detectaban erróneamente como en el caso del naranja y el rojo ya que, al ser colores similares, era difícil de controlar. Optamos por utilizar colores bastante diferentes entre sí y con rangos de color más concentrados.
- Dificultad para llevar datos por serial simultáneamente.
- Fallos por serial:
    - Colapso total de programas.
    - Pérdida de datos.
    - Problemas al asignar datos.
    - Manejo incorrecto de latencias entre Arduino y programa en Python con la frecuencia en la que se envían los datos.

7. CÓDIGO

Nuestro código se divide en dos partes, un código en el lenguaje Python para poder generar el programa que controla la detección de colores a través del uso de la librería OpenCV y un programa de Arduino con el cual podemos controlar directamente cómo queremos que actúen nuestros componentes físicos respectivamente con los datos suministrados por el programa en Python.

a. CÓDIGO ARDUINO

```
#include <Servo.h>


//Variable de Serial

String inputString = "";


//Variables de Pines

int buzzPin = 8;

int ledPin1 = 9;

int ledPin2 = 10;

int ledPin3 = 11;

int buttonPin = 2;

int redPin = A0;

int greenPin = A1;

int bluePin = A2;


//Variables Generales

int x;

int y;

int modo;


//Variables de Servos

Servo servoR;
```

```
Servo servoG;
Servo servoO;
Servo servoP;
Servo servoY;

//Variables de Estado de Servos (1 -> Abierto / 0 -> Cerrado)
int stateR = 1;
int stateG = 1;
int stateO = 1;
int stateP = 1;
int stateY = 1;

//Variables de Boton
int estadoBotonAnterior = HIGH;
unsigned long tiempoAnterior = 0;
const int debounceDelay = 50;  // Tiempo mínimo entre pulsaciones

void setup() {
  //Establece la finalidad de cada pin
  pinMode(buzzPin, OUTPUT);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(buttonPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
```

```
  //Establece parametros de la comunicacion Serial
  Serial.begin(115200);
  Serial.setTimeout(1);

  //Inicializa Variables
  x = 0;
  y = 0;
  modo = 3;

  //Establece Pines de Servos y los inicializa a velocidad 0 para evitar posibles activaciones al cargar el programa

  //servoO.attach(3);
  //servoO.write(90);
  servoR.attach(4);
  servoR.write(90);
  servoG.attach(5);
  servoG.write(90);
  servoP.attach(6);
  servoP.write(90);
  //servoY.attach(7);
  //servoY.write(90);
}

void loop() {

  //Lectura de Boton
```

```
 int estado = digitalRead(buttonPin);


 //Verifica si el botón se ha presionado sin rebotes
 if (estado == LOW && estadoBotonAnterior == HIGH && millis() - tiempoAnterior > debounceDelay) {
    //Cambia el modo. Reinicia si pasa de 3. Imprime el cambio por serial
    modo++;
    if (modo > 3) modo = 1;
    tiempoAnterior = millis();
    Serial.println("Modo cambiado a: " + String(modo));
 }


 // Guarda el estado anterior del boton
 estadoBotonAnterior = estado;


 if (modo == 1) {
  //MODO 1: Movimiento de Servos (Mano)
  digitalWrite(ledPin1, HIGH);
  digitalWrite(ledPin2, LOW);
  digitalWrite(ledPin3, LOW);


  //Variables de puntos medios de colores declarados
  float rx = 0;
  float ry = 0;


  float gx = 0;
  float gy = 0;
```

```
float bx = 0;
float by = 0;

float ox = 0;
float oy = 0;

float px = 0;
float py = 0;

float yx = 0;
float yy = 0;

//Mientras recibe datos por Serial
while (Serial.available()) {
  //Asigna los Valores de los puntos medios previamente declarados
  asignarValores(rx, ry, 'r');
  asignarValores(gx, gy, 'g');
  asignarValores(bx, by, 'b');
  asignarValores(ox, oy, 'o');
  asignarValores(px, py, 'p');
  asignarValores(yx, yy, 'y');

  //Mueve los Servos

  //moverServo(servoO, by, ox, oy, stateO, 100, 80, 800, 600);
  moverServo(servoR, by, rx, ry, stateR, 100, 80, 800, 600);
  moverServo(servoG, by, gx, gy, stateG, 100, 80, 800, 600);
  moverServo(servoP, by, px, py, stateP, 100, 80, 800, 600);
```

```
    //moverServo(servoY, by, yx, yy, stateY, 100, 80, 800, 600);
   }
  }
  else if (modo == 2) {
   //MODO 2: Potenciometro (Modulando sonido de Buzzer)
   digitalWrite(ledPin1, LOW);
   digitalWrite(ledPin2, HIGH);
   digitalWrite(ledPin3, LOW);

   //Lee los datos recogidos por serial
   //Importante, la manera de leer por serial en este apartado es distinta, ya que solo nos interesa capturar el punto rojo y su posición
   //Mientras recibe datos por Serial
   while (Serial.available()) {
    //Lee caracter a caracter hasta encontrar un salto de linea. Agrega a la cadena inputString estos caracteres
    char c = Serial.read();

    //Si llega al final, procesa la linea
    if (c == '\n') {
     //Limpia espacios o caracteres extra
     inputString.trim();

     //Obteniene el primer carácter limpio
     char tipoMensaje = inputString.charAt(0);

     //Delimita por las comas cada campo
     int primeraComa = inputString.indexOf(',');
     int segundaComa = inputString.indexOf(',', primeraComa + 1);
```

```
    if (primeraComa > 0 && segundaComa > primeraComa) {
      String x_str = inputString.substring(primeraComa + 1, segundaComa);
      String y_str = inputString.substring(segundaComa + 1);

      //Convierte cada elemento obtenido en cada campo en un float
      x = x_str.toFloat();
      y = y_str.toFloat();

      //Filtra que el color detefctado no sea nulo y solo sea rojo y ejecuta la funcion
      if (x != 0 && y != 0 && tipoMensaje == 'r') {
          reproducirSonido(x, y, buzzPin);
      }
     }
     inputString = "";  // Resetea la cadena para la siguiente lectura
   }
   //Si no llega al final, agrega el caracter
   else {
     inputString += c; // Sigue acumulando caracteres del mensaje
   }
  }
}
else if (modo == 3) {
  //MODO 3: Detección de colores combinados (Led RGB)
  digitalWrite(ledPin1, LOW);
  digitalWrite(ledPin2, LOW);
  digitalWrite(ledPin3, HIGH);
```

```
//Variables de puntos medios de colores declarados
float rx = 0;
float ry = 0;

float gx = 0;
float gy = 0;

float bx = 0;
float by = 0;

float ox = 0;
float oy = 0;

float px = 0;
float py = 0;

float yx = 0;
float yy = 0;

//Variables relativas al tiempo de activación del led RGB
unsigned long ultActivoR = 0;
unsigned long ultActivoG = 0;
unsigned long ultActivoB = 0;
int tiempoMinimo = 4000;

//Mientras recibe datos por Serial
while (Serial.available()) {
  //Asigna los Valores de los puntos medios previamente declarados
```

```
    asignarValores(rx, ry, 'r');
    asignarValores(gx, gy, 'g');
    asignarValores(bx, by, 'b');
    asignarValores(ox, oy, 'o');
    asignarValores(px, py, 'p');
    asignarValores(yx, yy, 'y');

    //Activa los leds. En un principio, deben estar activos un tiempo minimo
    if(rx > 0 && ry > 0){
      digitalWrite(redPin, HIGH);
      ultActivoR = millis();
    }else if(millis() - ultActivoR > tiempoMinimo){
      digitalWrite(redPin, LOW);
    }
    if(gx > 0 && gy > 0){
      digitalWrite(greenPin, HIGH);
      ultActivoG = millis();
    }else if(millis() - ultActivoG > tiempoMinimo){
      digitalWrite(greenPin, LOW);
    }
    if(bx > 0 && by > 0){
      digitalWrite(bluePin, HIGH);
      ultActivoB = millis();
    }else if(millis() - ultActivoB > tiempoMinimo){
      digitalWrite(bluePin, LOW);
    }
  }
}
```

```
}

void reproducirSonido(float xf, float yf, int buzzerPin){
 //Transforma Variables. Una simple regla de tres para determinar frecuencias(de 1000 a 5000) y volumenes(de 10 a 100) en funcion de posicion(de 20 a 620 o 450)
 xf = constrain(map(xf, 20, 620, 1000, 5000), 1000, 5000);
 yf = constrain(map(yf, 20, 450, 10, 100), 10, 100); //Ajuste para duty cycle

 //Reproduce Sonido con duty cycle

 //Duración del tono
 long duracionMs = 50;
 //Período de la frecuencia
 long periodMicrosec = 1000000L / xf;
 //Tiempo activo
 long highTime = (periodMicrosec * yf) / 100;
 //Tiempo inactivo
 long lowTime = periodMicrosec - highTime;
 //Numero de ciclos
 long ciclos = (xf * duracionMs) / 1000;

 //Por cada ciclo, durante su periodo de activacion, se hace sonar el buzzer pasivo a la frecuencia. Hacer esto nos permite modular el volumen
 for (long i = 0; i < ciclos; i++) {
    digitalWrite(buzzerPin, HIGH);
    delayMicroseconds(highTime);
    digitalWrite(buzzerPin, LOW);
    delayMicroseconds(lowTime);
 }
```

```
}

void asignarValores(float &cx, float &cy, char c) {
 //Los datos en este caso se pasan por referencia

 //Evita intentos de lectura sin datos
 if (!Serial.available()) return;

 //Lee el mensaje completo hasta salto de linea. Si el mensaje es vacio se ignora
 String mensaje = Serial.readStringUntil('\n');
 if (mensaje.length() == 0) return;

 //Delimita por las comas cada campo
 int primeraComa = mensaje.indexOf(',');
 int segundaComa = mensaje.indexOf(',', primeraComa + 1);

 if (primeraComa > 0 && segundaComa > primeraComa) {
  char tipoMensaje = mensaje[0];  // Extraer el primer carácter

  //Verifica el color correspondiente
  if (tipoMensaje == c) {
   String x_str = mensaje.substring(primeraComa + 1, segundaComa);
   String y_str = mensaje.substring(segundaComa + 1);

   //Convierte cada elemento obtenido en cada campo en un float
   cx = x_str.toFloat();
   cy = y_str.toFloat();
  }
```

```
  }
}

void moverServo(Servo miServo, float refY, float cx, float cy, int &state, int fuerzaCierre, int fuerzaApertura, int tiempoCierre, int tiempoApertura){

  //Si los puntos tomados como referencia no son nulos
  if (refY > 0 && cy > 0 && refY > cy){
    //Se calcula la diferencia entre la referencia de la base de la mano y la posicion de la y de el dedo dado
    int difference = refY - cy;
    //En caso de que la diferencia supere o no cierto rango, se asigna el estado pertinente
    int newState = (difference < 100) ? 0 : 1;

    //Solo cambiara si realmente hay un cambio
    if (newState != state) {

      //Se guarda el estado nuevo
      state = newState;

      //Si el el estado del servo es 1, se abre la palma, si es 0, se cierra
      //Importante: Si el servo es continuo (de 360 grados), se puede requerir delays y velocidades (fuerzas) distintos entre apertura y cierre
      //De ahi la necesida de pasar por parametros tiempos y velocidades distintos
      if (state == 1) {
        miServo.write(fuerzaApertura);
        delay(tiempoApertura);
      } else {
```

```
      miServo.write(fuerzaCierre);
      delay(tiempoCierre);
    }

    //Codigo Porisional. Objetivo: Mover los servos simultaneamente, y no en cascada, como sucede al usar delay()
    /*
      //Registro de tiempo para evitar delay
      unsigned long previousMillis = millis();
      //No bloquearia el programa completamente, permitiendo ejecución paralela
      while (millis() - previousMillis < 750) {
      }
    */
  }
  //Detiene el servo
  miServo.write(90);
 }
}
```

b. CÓDIGO EN PYTHON

```
import cv2
import numpy as np
import serial
import time
from PIL import Image
from collections import namedtuple

#Constantes de Colores
COLOR_RESALTE = (0, 0, 255)
```

```
# Captura de video desde la cámara del portátil
cap = cv2.VideoCapture(0)

# Definir el rango de color rojo en HSV
Color = namedtuple('Color', ['color_bajo_1', 'color_alto_1', 'color_bajo_2', 'color_alto_2'])

#[CONEXION ARDUINO]
#El diguiente fragmento conecta en serie con el arduino, en el puerto COM6
try:
  #arduino = serial.Serial(port='COM6', baudrate=9600, timeout=.1)
  arduino = serial.Serial(port='COM6', baudrate=115200, timeout=.1)
except serial.SerialException as e:
  print(f"Error Abriendo el puerto. Pruebe a cerrar la aplicacion de "
                f"Arduino o a conectar una laca al dispositivo: {e}")
  exit()

def envio_serial(arr, colCode):
  #Genera un String de a partir de los datos proporcionados. En caso de ser nulos, se igualan a 0
  if arr is not None:
    mensaje = f"{colCode}, {arr[4]}, {arr[5]}\n"
  else:
    mensaje = f"{colCode}, {0}, {0}\n"
  return mensaje

def definir_color(hsv_bajo1, hsv_alto1, hsv_bajo2, hsv_alto2):
  if all(len(arr) == 3 for arr in [hsv_bajo1, hsv_alto1, hsv_bajo2, hsv_alto2]):
```

```
      color = Color(np.array(hsv_bajo1), np.array(hsv_alto1), np.array(hsv_bajo2),
np.array(hsv_alto2))
      return color
   else:
      print('[DEBUG:FALLO_DE_PARAMETROS]')


def definir_mascara_color(color):
   mascara_color_1 = cv2.inRange(frame_hsv, color.color_bajo_1,
color.color_alto_1)
   mascara_color_2 = cv2.inRange(frame_hsv, color.color_bajo_2,
color.color_alto_2)
   mascara_color = mascara_color_1 + mascara_color_2


   #[TESTEO - BAJO PRUEBA DE RENDIMIENTO: Puede ser Elmininadosi baja
mucho el rendimiento]
   #Aplicar operaciones morfológicas para mejor detección de colores
   kernel = np.ones((5, 5), np.uint8)
   mascara_color = cv2.erode(mascara_color, kernel, iterations=1)
   mascara_color = cv2.dilate(mascara_color, kernel, iterations=1)


   return mascara_color


def definir_encapsulamiento_color(mascara_color, color, nombre_color):
   #Genera las coordenadas y su encapsulamiento
   #Genera los puntos que determinan el "Recipiente" de la mascara roja
   area_detectada = Image.fromarray(mascara_color).getbbox()
   if area_detectada is not None:
      #[INFO]
      #x1 = 0 esquina inferior izquierda
```

```
        #y1 = 0 esquina superior izquierda
        #x2 = 640 = esquina inferior derecha
        #y2 = 640 = esquina superior derecha
        x1, y1, x2, y2 = area_detectada
        media_x = int((x1 + x2)/2)
        media_y = int((y1 + y2)/2)

        color_bgr = cv2.cvtColor(np.uint8([[color.color_alto_1]]),
cv2.COLOR_HSV2BGR)[0][0]
        color_bgr = tuple(map(int, color_bgr))

        #Prints Auxiliares [Para Testear]
        #print(x1, y1, x2, y2)
        #print(mediaX, mediaY)
        recipiente= cv2.rectangle(resultado,
                          (x1, y1),
                          (x2, y2),
                          color_bgr,
                          5)
        punto_medio = cv2.circle(recipiente,
                         (media_x, media_y),
                         1,
                          COLOR_RESALTE,
                         5)

        cv2.putText(resultado, nombre_color, (x1, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
```

```
    return [x1, x2, y1, y2, media_x, media_y]

#region Sección Definición de Colores
rojo = definir_color([0, 180, 70],
                     [10, 255, 255],
                     [170, 180, 70],
                     [180, 255, 255])

verde = definir_color([35, 100, 50],
                      [85, 255, 255],
                      [35, 100, 50],
                      [85, 255, 255])

azul = definir_color([100, 150, 70],
                     [130, 255, 255],
                     [100, 150, 70],
                     [130, 255, 255])


amarillo = definir_color([25, 150, 100],
                         [35, 255, 255],
                         [25, 150, 100],
                         [35, 255, 255])


morado = definir_color([150, 40, 120],
                       [165, 255, 255],
                       [150, 40, 120],
```

```
                [165, 255, 255])
naranja = definir_color([5, 150, 200],
                [20, 255, 255],
                [5, 150, 200],
                [20, 255, 255])
#endregion

while True:
  # Leer el frame de la cámara. Ret devuelve el valor si ha tenido exito al leer la cámara
  ret, frame = cap.read()
  resultado = frame.copy()

  # Convertir el frame de BGR a HSV
  frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

  # Crear una máscara para el color rojo
  mascara_roja = definir_mascara_color(rojo)
  mascara_verde = definir_mascara_color(verde)
  mascara_azul = definir_mascara_color(azul)
  mascara_naranja = definir_mascara_color(naranja)
  mascara_morado = definir_mascara_color(morado)
  mascara_amarillo = definir_mascara_color(amarillo)

  # Aplicar la máscara al conjunto de mascaras coloreadas (solo para mostrar el resultado aislado)
  mascara_combinada = mascara_roja
  mascara_combinada = cv2.bitwise_or(mascara_combinada, mascara_verde)
  mascara_combinada = cv2.bitwise_or(mascara_combinada, mascara_azul)
```

```
    mascara_combinada = cv2.bitwise_or(mascara_combinada, mascara_naranja)
    mascara_combinada = cv2.bitwise_or(mascara_combinada, mascara_morado)
    mascara_combinada = cv2.bitwise_or(mascara_combinada, mascara_amarillo)
    mascara_coloreada = cv2.bitwise_and(frame, frame,
mask=mascara_combinada)

    #Definición de encapsulamiento de las máscaras
    arr_Ro = definir_encapsulamiento_color(mascara_roja, rojo, "rojo")
    arr_Ve = definir_encapsulamiento_color(mascara_verde, verde, "verde")
    arr_Az = definir_encapsulamiento_color(mascara_azul, azul, "azul")
    arr_Na = definir_encapsulamiento_color(mascara_naranja, naranja, "naranja")
    arr_Mo = definir_encapsulamiento_color(mascara_morado, morado, "morado")
    arr_Am = definir_encapsulamiento_color(mascara_amarillo, amarillo, "amarillo")

    #Se genera la hilera de datos que enviaremos por Serial a Arduino
    mensajeSerial = ""
    mensajeSerial += envio_serial(arr_Ro, 'r')
    mensajeSerial += envio_serial(arr_Ve, 'g')
    mensajeSerial += envio_serial(arr_Az, 'b')
    mensajeSerial += envio_serial(arr_Na, 'o')
    mensajeSerial += envio_serial(arr_Mo, 'p')
    mensajeSerial += envio_serial(arr_Am, 'y')

    # Esta funcion se encarga de comunicarse con el arduino. Envía por Serial una
secuencia de bytes a la Placa Arduino
    # Luego, si hay una respues por Serial por parte de arduino, se captura y se
muestra
    if mensajeSerial:
        arduino.write(mensajeSerial.encode())
```

```
        time.sleep(0.05)
        if arduino.in_waiting > 0:
            data = arduino.readline()
            print(data)

    #Dibuja un punto en el medio del resultado
    cv2.circle(resultado,
               (320, 320),
               1,
               (0, 0, 255),
               5)

    # Mostrar el frame original, la máscara y el resultado
    cv2.imshow('Video Original', frame)
    cv2.imshow('Mascara Coloreada', mascara_coloreada)
    cv2.imshow('Resultado', resultado)

    # Salir del bucle con la tecla 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Liberar la cámara y cerrar todas las ventanas
cap.release()
cv2.destroyAllWindows()
```

8. CASOS DE USO

Nuestro proyecto consta esencialmente de tres modos para los cuales, una vez cargado nuestro programa en Arduino a la placa, debemos ejecutar en todos los casos el código en Python:

- Para el caso de la mano, siendo este el caso principal, utilizando el color azul en la palma de nuestra mano real y mostrarla en la cámara, podemos realizar distintos movimientos con nuestros dedos teniendo el color rojo en el dedo índice, el verde en el corazón y el morado en el anular. Si nosotros cerramos dichos dedos, la réplica se moverá consecutivamente y si los abrimos, la réplica hará lo mismo.
- Para el caso de la réplica de un potenciómetro, nuestro buzzer reacciona en función de los movimientos del color rojo en la cámara. Es decir, si nosotros movemos el color rojo de arriba a abajo nuestro buzzer sonará más alto o bajo y si nosotros lo movemos de izquierda a derecha este cambiará el tono sonando más agudo o grave.
- Para el último caso del led, funciona con los 3 colores primarios. Si nosotros mostramos el color rojo, el led rojo del RGB se encenderá y con el azul y el verde se encenderán sus leds correspondientes. Con esto podemos crear los efectos visuales de las distintas mezclas, obteniendo el amarillo con el rojo y el verde, el cyan con el azul y el verde, el magenta con el rojo y azul y finalmente el blanco si mostramos los 3 a la vez.

9. POSIBLES MEJORAS FUTURAS

Además, nos gustaría tener en cuenta unas posibles mejoras a futuro para mejorar el proyecto o llevarlo a otro nivel de escala:

- Cambiar el sistema de servomotores, siendo así servomotores posicionales y no continuos.
- Cambiar el sistema de contrapeso.
- Mejorar calidad de materiales.
- Añadir el dedo pulgar y el meñique.
- Afinar el rango de colores.
- Evitar parpadeos en la luz RGB.

Finalmente, el más importante, sería la opción de empotrar el sistema con una Raspberry pi 3 modelo b y una cámara web externa. Realizamos pruebas con la misma, pero no creímos que fuese lo suficientemente manejable en el aula debido a que era necesario un monitor externo.