**RobotiArm Classifier**

**Reporte de Brazo robótico 5DOF**

**Autor: Elías José Jara Pinto**

**1. Introducción**

En la era de la automatización y la inteligencia artificial, la integración de sistemas robóticos con capacidades de visión por computadora ha revolucionado diversas industrias. Este proyecto se centra en el desarrollo de un **brazo robótico de 5 grados de libertad (5DOF)**, diseñado para clasificar piezas basadas en números impresos en ellas. Utilizando un **Arduino Uno/Nano** para el control de servomotores y **Python** para la visión y la interfaz gráfica, el sistema identifica el número en una pieza y la posiciona automáticamente en una de cuatro ubicaciones predefinidas.

**2. Objetivos del Proyecto**

**2.1 Objetivo Principal**

Desarrollar un brazo robótico autónomo capaz de identificar números escritos en piezas mediante visión por computadora y clasificarlas automáticamente en una de cuatro posiciones fijas.

**2.2 Objetivos Específicos**

* **Diseño Mecánico:** Crear y ensamblar la estructura del brazo robótico utilizando piezas impresas en 3D (PLA) con 5 grados de libertad.
* **Control Electrónico:** Integrar y controlar 4 servomotores MG996R y 2 servomotores MG90S mediante un Arduino Uno/Nano.
* **Visión por Computadora:** Implementar un sistema de reconocimiento de números utilizando Python y TensorFlow/Keras.
* **Interfaz de Usuario:** Desarrollar una interfaz gráfica en Python (Tkinter) para el control manual y la supervisión del sistema.
* **Integración de Sistema:** Combinar los componentes mecánicos, electrónicos y de software para lograr una operación autónoma.
* **Pruebas y Validación:** Realizar pruebas exhaustivas para asegurar la precisión en la clasificación y el funcionamiento correcto del brazo.

**3. Descripción del Robot y su Mecanismo**

El brazo robótico está diseñado con **5 Grados de Libertad (DOF)**, excluyendo el movimiento de cierre y apertura del gripper. Este diseño permite que el efector final (gripper) alcance y oriente objetos en cualquier posición dentro del espacio de trabajo, proporcionando flexibilidad y funcionalidad. A continuación, se detalla cada uno de los componentes y mecanismos relacionados con los DOF, así como las articulaciones necesarias para lograr estos movimientos.

**3.1 Determinación de los Grados de Libertad (DOF)**

Los **5 DOF** del brazo robótico corresponden a los siguientes movimientos y articulaciones:

|  |  |  |  |
| --- | --- | --- | --- |
| **DOF** | **Movimiento** | **Articulación** | **Servomotor** |
| 1 | Rotación de la Base (Cintura) | Base-Plato Giratorio | MG996R |
| 2 | Movimiento del Hombro | Plato-Eslabón 1 | MG996R |
| 3 | Movimiento del Codo | Eslabón 1-Eslabón 2 | MG996R |
| 4 | Inclinación del Gripper (Cabeceo) | Eslabón 2-Eslabón 3 | MG996R |
| 5 | Rotación del Gripper (Rodamiento) | Eslabón 3-Gripper | MG90S |

**Nota:** El movimiento de apertura y cierre del gripper es controlado de manera independiente y no se considera un DOF adicional.

**3.2 Descripción Detallada de Cada DOF**

**DOF 1 - Rotación de la Base (Cintura)**

* **Controlado por:** Servomotor MG996R.
* **Función:** Permite que el brazo gire sobre su eje vertical, posicionándose en cualquier dirección horizontal.
* **Mecanismo:** Se logra mediante la rotación del plato giratorio montado sobre la base.

Imagen que contiene avión, aire, cocina, hombre

Descripción generada automáticamente

Figura . Unión de la Base con el Plato giratorio.

**DOF 2 - Movimiento del Hombro**

* **Controlado por:** Servomotor MG996R montado en el Plato Giratorio.
* **Función:** Proporciona movimiento hacia arriba y hacia abajo del brazo principal, ajustando su altura.
* **Importancia:** Esencial para alcanzar diferentes planos de trabajo.

Imagen que contiene Diagrama

Descripción generada automáticamente

Figura . Unión del Plato con el Eslabón 1.

**DOF 3 - Movimiento del Codo**

* **Controlado por:** Servomotor MG996R montado en el Eslabón 1.
* **Función:** Permite que el brazo se flexione o extienda, aumentando el rango de alcance del gripper.
* **Características Especiales:**
  + **Orientación Contraria:** El servomotor que controla el DOF 3 está orientado de forma contraria a los servos que controlan el DOF 2 y DOF 4, es decir, gira positivamente en dirección contraria.

Imagen que contiene celular

Descripción generada automáticamente

Figura . Unión del Plato y Eslabón 1 con el Eslabón 2.

**DOF 4 - Inclinación del Gripper (Cabeceo)**

* **Controlado por:** Servomotor MG996R montado en el Eslabón 2.
* **Función:** Proporciona la capacidad de inclinar el gripper hacia arriba o hacia abajo.
* **Importancia:** Esencial para trabajar con superficies inclinadas o ajustar la orientación del objeto sujetado.

Dibujo de una persona

Descripción generada automáticamente con confianza baja

Figura . Unión del Plato, Eslabón 1 y Eslabón 2 con el Eslabón 3.

**DOF 5 - Rotación del Gripper (Rodamiento)**

* **Controlado por:** Servomotor MG90S montado en el Eslabón 3.
* **Función:** Permite que el gripper gire sobre su eje longitudinal, reorientando los objetos manipulados.
* **Importancia:** Facilita la orientación de los objetos en posiciones específicas sin mover el resto del brazo.

Diagrama

Descripción generada automáticamente

Figura . Unión del Gripper con el Eslabón 3.

**3.3 Gripper (Garra)**

Aunque no se considera un DOF adicional, el gripper incluye un movimiento funcional independiente para abrir y cerrar, controlado por un servomotor MG90S. Este movimiento es fundamental para sujetar y liberar objetos con precisión y no afecta la orientación ni la posición del gripper.

* **Movimiento de Apertura y Cierre:**
  + **Controlado por:** Servomotor MG90S.
  + **Función:** Permite sujetar y liberar objetos con precisión.

Imagen que contiene lego, juguete

Descripción generada automáticamente

Figura . Mecanismo del Gripper.

**4. Diseño y Ensamblaje Mecánico**

**4.1 Estructura Mecánica**

La estructura mecánica del brazo robótico está compuesta por una serie de eslabones y componentes diseñados para soportar los servomotores y permitir los movimientos definidos por los DOF. Todas las piezas principales están fabricadas en **PLA mediante impresión 3D**, lo que proporciona ligereza y resistencia.

**4.2 Componentes Principales**

**4.2.1 Base y Plato Giratorio (DOF 1)**

* **Base:**
  + **Material:** PLA impreso en 3D.
  + **Función:** Proporciona estabilidad al sistema.
  + **Características:**
    - Incluye un rodamiento en el eje central para facilitar la rotación suave del plato giratorio.
    - Diseñada para soportar el peso y el torque de los servos sin deformarse.
* **Plato Giratorio:**
  + **Material:** PLA impreso en 3D.
  + **Función:** Conecta la base con el Eslabón 1.
  + **Características:**
    - Soporta el servomotor MG996R que controla el movimiento del Eslabón 1 (DOF 2).
    - Permite un giro de hasta 180°, ofreciendo una amplia gama de movimiento horizontal.

**4.2.2 Eslabones (DOF 2, DOF 3 y DOF 4)**

* **Eslabón 1 (DOF 2 y DOF 3):**
  + **Material:** PLA impreso en 3D.
  + **Función:** Une el Plato Giratorio con el Eslabón 2.
  + **Características:**
    - Montaje del servomotor MG996R que controla el movimiento vertical del Eslabón 2 (DOF 3).
    - Diseño que soporta la orientación contraria del servomotor DOF 3 respecto a DOF 2 y DOF 4.
* **Eslabón 2 (DOF 4):**
  + **Material:** PLA impreso en 3D.
  + **Función:** Conecta el Eslabón 1 con el Eslabón 3.
  + **Características:**
    - Montaje del servomotor MG996R que controla la inclinación del gripper (DOF 4).
    - Diseñado para soportar el torque del servomotor sin comprometer la estabilidad.

**4.2.3 Eslabón 3 y Gripper (DOF 5)**

* **Eslabón 3 (DOF 5):**
  + **Material:** PLA impreso en 3D.
  + **Función:** Conecta el Eslabón 2 con el Gripper.
  + **Características:**
    - Soporta dos servomotores MG90S:
      * **Servomotor para Rodamiento:** Controla la rotación del gripper (DOF 5).
      * **Servomotor para Apertura/Cierre:** Controla la apertura y cierre del gripper.
    - Diseño robusto que transmite eficientemente el torque de los servos al gripper sin comprometer la estabilidad del brazo.
* **Gripper (Garra):**
  + **Material:** PLA impreso en 3D.
  + **Función:** Sujetar y liberar objetos con precisión.
  + **Características:**
    - Diseño ergonómico con dos mordazas accionadas por el servomotor MG90S.
    - Movimiento independiente para abrir y cerrar las mordazas, permitiendo una sujeción firme y controlada.

**4.4 Ensamblaje Mecánico**

**4.4.1 Ensamblaje de la Base y Plato Giratorio (DOF 1)**

1. **Instalación del Rodamiento:**
   * Colocación del rodamiento en el alojamiento central de la base.
   * Asegurar que el plato giratorio gire suavemente sin juego excesivo.
2. **Montaje del Servomotor MG996R:**
   * Fijación del servomotor MG996R en el plato giratorio.
   * Conexión del eje del servomotor al plato para permitir la rotación controlada.

**4.4.2 Ensamblaje del Eslabón 1 (DOF 2 y DOF 3)**

1. **Montaje del Servomotor MG996R (DOF 2):**
   * Fijación del servomotor MG996R en el plato giratorio.
   * Conexión del servomotor al Eslabón 1 para controlar el movimiento vertical (DOF 2).
2. **Orientación Contraria del Servomotor DOF 3:**
   * Instalación del servomotor MG996R en el Eslabón 1, orientado de forma contraria respecto a los servos DOF 2 y DOF 4.
   * Esto permite un movimiento de flexión/extensión (DOF 3) más preciso y controlado.

**4.4.3 Ensamblaje del Eslabón 2 (DOF 4)**

1. **Montaje del Servomotor MG996R:**
   * Fijación del servomotor MG996R en el Eslabón 2.
   * Conexión del servomotor al Eslabón 3 para controlar la inclinación del gripper (DOF 4).
2. **Diseño Robusto:**
   * Asegurar que el Eslabón 2 pueda soportar el torque del servomotor sin deformarse.
   * Verificación de la estabilidad del movimiento de inclinación.

**4.4.4 Ensamblaje del Eslabón 3 y Gripper (DOF 5)**

1. **Montaje de los Servomotores MG90S:**
   * Fijación de los servomotores MG90S en el Eslabón 3.
     + **Servomotor para Rodamiento:** Controla la rotación del gripper (DOF 5).
     + **Servomotor para Apertura/Cierre:** Controla la apertura y cierre del gripper.
2. **Montaje del Gripper:**
   * Conexión del gripper al Eslabón 3.
   * Asegurar que las mordazas del gripper se muevan de manera independiente y suave.

**4.5 Pruebas Mecánicas Iniciales**

* **Movimientos Manuales:**
  + Manipulación manual de cada articulación para verificar la libertad de movimiento.
  + Asegurar que no existan obstrucciones ni fricciones excesivas.
* **Equilibrio del Brazo:**
  + Verificación de que el brazo no se incline hacia adelante o hacia atrás debido al peso.
  + Adición de contrapesos si es necesario para mantener el equilibrio.



Figura . Brazo robótico complemente ensamblado.

**5. Cinemática del robot**

A continuación, se presenta la cinemática directa e inversa del brazo robótico con 5 Grados de Libertad (5 DOF) descrito.

**5.1 Cinemática Directa**

**1. Revisión de los Parámetros D-H y las Matrices ​**

La tabla de parámetros D-H proporcionada es la siguiente:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Eslabón** |  |  |  |  |
| 1 |  | 12 | 0 | +90° |
| 2 | ​ | 0 | 15 | 0° |
| 3 |  | 0 | 10 | 0° |
| 4 |  | 0 | 0 | -90° |
| 5 | ​ | 18.1 | 0 | 0° |

La matriz D-H genérica es:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Matriz**   
Parámetros: ​, , , .

Texto

Descripción generada automáticamente con confianza media

**Matriz**

Parámetros: ​, , , .

Texto

Descripción generada automáticamente

**Matriz**

Parámetros: ​, , , .

Texto

Descripción generada automáticamente

**Matriz**

Parámetros: ​, , , .

Imagen que contiene Texto

Descripción generada automáticamente

**Matriz**

Parámetros: ​, , , .

Imagen de la pantalla de un celular de un mensaje en letras negras

Descripción generada automáticamente con confianza baja

**Matriz de Transformación Final**

Para el resultado final de la Posición (dx, dy, dz) y Orientación



Texto

Descripción generada automáticamente

Se obtiene los siguientes coeficientes:

Texto

Descripción generada automáticamente

Se obtienen las siguientes posiciones:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**5.2 Cinemática Inversa**

La cinemática inversa (IK) consiste en encontrar los ángulos de las articulaciones ​ a partir de una posición y orientación deseadas del efector final. El procedimiento típico involucra:

1. **Desacople de la muñeca**: Separar el problema en dos partes. Primero se resuelven los tres primeros grados de libertad (generalmente asociados a la posición del centro de la muñeca), y luego se calculan los últimos dos grados de libertad (asociados a la orientación final).
2. **Cálculo del Centro de la Muñeca (Wrist Center)**:  
   A partir de la posición deseada del efector y la orientación final (que nos provee el vector director del eje z del efector, dado por , se calcula la posición del centro de la muñeca restando la componente a lo largo del último eje:



Se asume que el último eslabón (con longitud ​) se extiende a lo largo del eje z del efector final. Al "retroceder" a lo largo de este eje z en el espacio del efector, se obtiene la posición del punto donde inician las últimas articulaciones de la muñeca. Este paso es habitual en robótica, especialmente en robots con “wrist” (muñeca) compuesto por los últimos 2 ó 3 DOF.

1. **Determinación de ​**:  
   Conocido el centro de la muñeca , se resuelve el subproblema de un manipulador de 3 grados de libertad (los primeros 3 DOF), que normalmente definen la posición del extremo antes de la muñeca.
   * **Cálculo de ​**:

El primer ángulo es la rotación de la base alrededor del eje z, y se determina sencillamente a partir de la proyección del punto objetivo sobre el plano XY.

* + **Cálculo de ​** (el codo) y (el hombro):

Imagen de la pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza baja

Esta expresión surge de la aplicación de la Ley de Cosenos en el plano formado por los eslabones 2 y 3. Aquí se asume:

* + - El offset vertical inicial es de 12 cm (asociado al primer eslabón).
    - La longitud del segundo eslabón es 15 cm.
    - La longitud del tercer eslabón es 10 cm.

La Ley de Cosenos para encontrar el ángulo en la articulación del codo (​) es una práctica estándar en cinemática inversa para robots con geometría tipo RR (rotacional-rotacional en el plano). Dicho de forma genérica:

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

donde R es la distancia desde el eje del hombro hasta el centro de la muñeca.

D está definido de una manera ligeramente diferente, pero es equivalente a la hora de recuperar ​.

Una vez calculado ​, para ​ se utilizan relaciones trigonométricas adicionales:



Esta es una fórmula estándar derivada del análisis geométrico del brazo para ubicar el hombro. Se ve compleja, pero es una forma conocida de resolver el ángulo del hombro después de haber determinado el ángulo del codo.

Con esto se obtienen ​.

1. **Cálculo de ​ y ​**:  
   Una vez se tienen los primeros 3 ángulos, se puede calcular la orientación de la muñeca aislando la rotación final. Esto se hace con:



donde es la orientación deseada final del efector, ​ es la orientación de la base hasta la articulación 3 (calculada a partir de ​) y ​ es la rotación pura debida a las articulaciones 4 y 5.

Texto, Carta

Descripción generada automáticamente

Estas fórmulas para ​ y son algo complejas​. Esto se debe a que el robot no posee una “muñeca esférica” estándar (3 DOF en la muñeca), sino solo 2 DOF adicionales. Esto obliga a un análisis más detallado para extraer ​ a partir de la submatriz .

Primero se obtiene la orientación parcial generada por los primeros 3 ejes, luego se compara con la orientación final requerida para la herramienta. De allí se despejan las rotaciones faltantes, ​ y .

La complejidad de las ecuaciones es normal cuando la muñeca no es esférica. En manipuladores con muñecas menos simétricas, las ecuaciones para extraer ángulos se complican.

**6. Selección y Configuración de Hardware Electrónico**

**6.1 Lista de Componentes**

* **Microcontrolador:** Arduino Uno o Arduino Nano.
* **Servomotores:**
  + **4 x MG996R:** Para los DOF 1, 2, 3 y 4.
  + **2 x MG90S:**
    - **Servomotor MG90S #1:** Para la rotación del gripper (DOF 5).
    - **Servomotor MG90S #2:** Para la apertura y cierre del gripper.
* **Fuente de Alimentación:**
  + Adaptador de 5V y al menos 15A para alimentar de forma estable los servos.
* **Cables y Conectores:**
  + Cables jumper macho-macho y hembra-hembra.
  + Conectores específicos para servos.
* **Protoboard:**
  + Para organizar y distribuir la alimentación y señales.
* **Otros Componentes:**
  + Condensadores de desacoplo (100µF) para estabilizar la alimentación.

**6.2 Conexión de Servomotores**

**6.2.1 Alimentación de los Servos**

* **Alimentación Externa:**
  + **Cables Rojos (VCC):** Conectar a la línea positiva de la fuente de alimentación de 5V.
  + **Cables Negros/Marrones (GND):** Conectar a la línea de tierra común.
* **Importante:**
  + **Fuente Externa:** Nunca alimentar los servos directamente desde el Arduino para evitar sobrecargar el microcontrolador.
  + **Conexión de Tierra Común:** Asegurar que la tierra (GND) de la fuente de alimentación externa esté conectada al GND del Arduino.

**6.2.2 Conexión de Señales de Control**

**Cable naranja:** Asignación de pines del Arduino para cada servomotor:

|  |  |  |
| --- | --- | --- |
| **Servomotor** | **Movimiento** | **Pin Arduino** |
| MG996R #1 | Rotación de la Base (DOF 1) | 3 |
| MG996R #2 | Movimiento del Hombro (DOF 2) | 5 |
| MG996R #3 | Movimiento del Codo (DOF 3) | 6 |
| MG996R #4 | Inclinación del Gripper (DOF 4) | 9 |
| MG90S #1 | Rotación del Gripper (DOF 5) | 10 |
| MG90S #2 | Apertura/Cierre del Gripper | 11 |

**Nota:** Asegurar que cada servomotor está correctamente asignado al pin correspondiente y que los cables naranjas están firmemente conectados.

**6.2.3 Configuración de los Servos**

* **Posiciones Iniciales:**
  + **MG996R #1 (DOF 1):** 180°
  + **MG996R #2 (DOF 2):** 165°
  + **MG996R #3 (DOF 3):** 90°
  + **MG996R #4 (DOF 4):** 90°
  + **MG90S #1 (DOF 5):** 90°
  + **MG90S #2 (Gripper):** 130°
* **Importancia de las Posiciones Iniciales:**
  + Aseguran que el brazo comienza en una posición estable y centralizada.
  + Facilitan la calibración y el control posterior del brazo.

**6.3 Configuración del Arduino Uno/Nano**

**6.3.1 Limitaciones de Pines**

* **Arduino Uno/Nano:**
  + Ambos modelos cuentan con suficientes pines digitales para controlar los servos.
  + La librería **Servo.h** permite controlar servos desde cualquier pin digital, aunque en algunos casos puede afectar funciones temporizadas.

**6.3.2 Uso de Librerías**

* **Servo.h:**
  + Facilita el control de servomotores conectados a los pines designados.
  + Permite definir ángulos específicos para cada servo de manera sencilla.

**6.4 Fuente de Alimentación**

**6.4.1 Cálculo del Consumo**

* **Servomotores MG996R:**
  + Cada servomotor puede consumir hasta 2.5A en carga máxima.
  + **4 x MG996R:** 4 x 2.5A = 10A
* **Servomotores MG90S:**
  + Cada servomotor puede consumir hasta 1A en carga máxima.
  + **2 x MG90S:** 2 x 1A = 2A
* **Total Máximo:**
  + **12A**

**6.4.2 Selección de la Fuente**

* **Fuente Recomendada:**
  + Adaptador de 5V y al menos 15A para proporcionar un margen de seguridad y evitar caídas de voltaje.
* **Alternativa:**
  + Utilizar dos fuentes separadas:
    - **Una para el Arduino:** Alimentada por USB o un adaptador de 5V de baja corriente.
    - **Otra para los Servos:** Fuente de 5V y 15A conectada directamente a los servos.

**7. Implementación del Software**

El software del proyecto está dividido en varias partes que manejan el control de servomotores, la interfaz gráfica, la visión por computadora y la lógica de clasificación. A continuación, se detalla cada componente.

**7.1 Firmware del Arduino**

**7.1.1 Descripción General**

El firmware se encarga de recibir comandos desde el PC a través de la comunicación serial y controlar los servomotores en consecuencia. Reconoce comandos para mover los servos a posiciones específicas y para abrir/cerrar el gripper.

**7.1.2 Código Fuente:** arduino\_test.ino

1. // arduino\_test.ino

2.

3. #include <Servo.h>

4.

5. // Definir los objetos Servo para cada uno de los 6 servos

6. Servo servo1, servo2, servo3, servo4, servo5, servo6;

7.

8. // Asignar los pines a cada servo

9. const int pinServo1 = 3;

10. const int pinServo2 = 5;

11. const int pinServo3 = 6;

12. const int pinServo4 = 9;

13. const int pinServo5 = 10;

14. const int pinServo6 = 11;

15.

16. void setup() {

17. // Iniciar la comunicación serial a 9600 bps

18. Serial.begin(9600);

19.

20. // Adjuntar cada servo a su pin correspondiente

21. servo1.attach(pinServo1);

22. servo2.attach(pinServo2);

23. servo3.attach(pinServo3);

24. servo4.attach(pinServo4);

25. servo5.attach(pinServo5);

26. servo6.attach(pinServo6);

27.

28. // Posición inicial para todos los servos

29. servo1.write(180);

30. servo2.write(165);

31. servo3.write(90);

32. servo4.write(90);

33. servo5.write(90);

34. servo6.write(130);

35.

36. // Confirmar inicio

37. Serial.println("Arduino listo para recibir comandos.");

38. }

39.

40. void loop() {

41. // Verificar si hay datos disponibles en el buffer serial

42. if (Serial.available() > 0) {

43. // Leer la línea completa hasta el carácter de nueva línea

44. String command = Serial.readStringUntil('\n');

45. command.trim(); // Eliminar espacios en blanco adicionales

46.

47. // Comandos esperados:

48. // "MOVE t1 t2 t3 t4 t5 t6"

49. // "GRIP OPEN"

50. // "GRIP CLOSE"

51.

52. if (command.startsWith("MOVE")) {

53. // Dividir el comando por espacios

54. int t1, t2, t3, t4, t5, t6;

55. sscanf(command.c\_str(), "MOVE %d %d %d %d %d %d", &t1, &t2, &t3, &t4, &t5, &t6);

56.

57. // Limitar los ángulos entre 0 y 180 grados

58. t1 = constrain(t1, 0, 180);

59. t2 = constrain(t2, 0, 180);

60. t3 = constrain(t3, 0, 180);

61. t4 = constrain(t4, 0, 180);

62. t5 = constrain(t5, 0, 180);

63. t6 = constrain(t6, 0, 180);

64.

65. // Mover cada servo a su ángulo correspondiente

66. servo1.write(t1);

67. servo2.write(t2);

68. servo3.write(t3);

69. servo4.write(t4);

70. servo5.write(t5);

71. servo6.write(t6);

72.

73. // Confirmar movimiento

74. Serial.println("MOVIMIENTO COMPLETADO");

75. }

76. else if (command == "GRIP OPEN") {

77. // Abrir gripper (servo6 a 180 grados, por ejemplo)

78. servo6.write(180);

79. Serial.println("GRIPPER ABIERTOS");

80. }

81. else if (command == "GRIP CLOSE") {

82. // Cerrar gripper (servo6 a 130 grados, por ejemplo)

83. servo6.write(130);

84. Serial.println("GRIPPER CERRADOS");

85. }

86. else {

87. // Comando no reconocido

88. Serial.println("COMANDO NO RECONOCIDO");

89. }

90. }

91. }

92.

**7.1.3 Explicación del Código**

* **Inicialización:**
  + Se definen seis objetos Servo para controlar cada servomotor.
  + Cada servo se adjunta a un pin específico del Arduino.
  + Los servos se posicionan inicialmente en ángulos predeterminados.
* **Recepción de Comandos:**
  + El Arduino escucha continuamente el puerto serial.
  + Reconoce tres tipos de comandos:
    - **MOVE:** Para mover los seis servos a los ángulos especificados.
    - **GRIP OPEN:** Para abrir el gripper.
    - **GRIP CLOSE:** Para cerrar el gripper.
  + Después de ejecutar un comando, el Arduino envía una confirmación de vuelta al PC.

**7.2 Control Manual y GUI**

**7.2.1 Descripción General**

El control manual del brazo se implementa mediante una interfaz gráfica desarrollada con Tkinter en Python. Esta interfaz permite al usuario ajustar los ángulos de los servos utilizando sliders y controlar la apertura/cierre del gripper mediante botones.

**7.2.2 Código Fuente: test\_control.py**

1. # test\_control.py

2.

3. import serial

4. import time

5. import tkinter as tk

6. from tkinter import ttk, messagebox

7. import threading

8.

9.

10. class ServoController:

11. def \_\_init\_\_(self, port='COM6', baudrate=9600, timeout=1):

12. try:

13. self.ser = serial.Serial(port, baudrate, timeout=timeout)

14. time.sleep(2) # Esperar a que el Arduino reinicie

15. print(f"Conectado a {port} a {baudrate} bps.")

16. self.read\_serial() # Leer el mensaje de inicio

17. except serial.SerialException as e:

18. print(f"Error al conectar con el puerto serial: {e}")

19. self.ser = None

20.

21. def send\_command(self, command):

22. if self.ser and self.ser.is\_open:

23. try:

24. self.ser.write((command + '\n').encode())

25. print(f"Enviado: {command}")

26. # Leer respuesta no bloqueante

27. response = self.read\_serial()

28. return response

29. except serial.SerialException as e:

30. print(f"Error al enviar comando: {e}")

31. return None

32. else:

33. print("Puerto serial no está abierto.")

34. return None

35.

36. def read\_serial(self):

37. if self.ser and self.ser.in\_waiting > 0:

38. try:

39. response = self.ser.readline().decode().strip()

40. if response:

41. print(f"Arduino: {response}")

42. return response

43. except serial.SerialException as e:

44. print(f"Error al leer del serial: {e}")

45. return None

46.

47. def move\_servos(self, angles):

48. """

49. angles: lista o tupla de 6 ángulos (t1, t2, t3, t4, t5, t6)

50. """

51. if len(angles) != 6:

52. print("Debe proporcionar exactamente 6 ángulos.")

53. return None

54. command = f"MOVE {' '.join(map(str, angles))}"

55. return self.send\_command(command)

56.

57. def grip\_open(self):

58. return self.send\_command("GRIP OPEN")

59.

60. def grip\_close(self):

61. return self.send\_command("GRIP CLOSE")

62.

63. def close(self):

64. if self.ser and self.ser.is\_open:

65. self.ser.close()

66. print("Puerto serial cerrado.")

67.

68.

69. class ServoGUI:

70. def \_\_init\_\_(self, root, controller: ServoController):

71. self.root = root

72. self.controller = controller

73. self.root.title("Control de Servos con Arduino")

74. self.create\_widgets()

75. # self.current\_angles = [90] \* 6 # Ángulos iniciales

76. self.current\_angles = [180,165, 90, 90, 90, 180]

77. self.update\_log\_thread = threading.Thread(target=self.update\_log, daemon=True)

78. self.update\_log\_thread.start()

79.

80. def create\_widgets(self):

81. # Crear un frame para los sliders

82. sliders\_frame = ttk.LabelFrame(self.root, text="Control de Servos")

83. sliders\_frame.pack(padx=10, pady=10, fill="x")

84.

85. self.sliders = []

86. self.slider\_vars = []

87. self.angle\_labels = []

88.

89. for i in range(6):

90. var = tk.IntVar(value=90)

91. self.slider\_vars.append(var)

92. slider = ttk.Scale(

93. sliders\_frame,

94. from\_=0,

95. to=180,

96. orient='horizontal',

97. variable=var,

98. command=lambda val, idx=i: self.update\_label(idx)

99. )

100. slider.grid(row=i, column=1, padx=5, pady=5, sticky="ew")

101. sliders\_frame.columnconfigure(1, weight=1)

102.

103. label = ttk.Label(sliders\_frame, text=f"Servo {i + 1}:")

104. label.grid(row=i, column=0, padx=5, pady=5, sticky="w")

105.

106. angle\_label = ttk.Label(sliders\_frame, text="90")

107. angle\_label.grid(row=i, column=2, padx=5, pady=5, sticky="w")

108. self.sliders.append(slider)

109. self.angle\_labels.append(angle\_label)

110.

111. # Botón para mover el brazo

112. move\_button = ttk.Button(self.root, text="Mover Servos", command=self.move\_servos\_smooth)

113. move\_button.pack(pady=5)

114.

115. # Botones para abrir y cerrar el gripper

116. gripper\_frame = ttk.Frame(self.root)

117. gripper\_frame.pack(pady=5)

118.

119. open\_button = ttk.Button(gripper\_frame, text="Abrir Gripper", command=self.open\_gripper)

120. open\_button.grid(row=0, column=0, padx=5)

121.

122. close\_button = ttk.Button(gripper\_frame, text="Cerrar Gripper", command=self.close\_gripper)

123. close\_button.grid(row=0, column=1, padx=5)

124.

125. # Área de log para mostrar respuestas del Arduino

126. log\_frame = ttk.LabelFrame(self.root, text="Log de Comunicación")

127. log\_frame.pack(padx=10, pady=10, fill="both", expand=True)

128.

129. self.log\_text = tk.Text(log\_frame, height=10, state='disabled')

130. self.log\_text.pack(padx=5, pady=5, fill="both", expand=True)

131.

132. # Botón para salir

133. exit\_button = ttk.Button(self.root, text="Salir", command=self.on\_exit)

134. exit\_button.pack(pady=5)

135.

136. def update\_label(self, idx):

137. angle = self.slider\_vars[idx].get()

138. self.angle\_labels[idx].config(text=str(int(angle)))

139.

140. def move\_servos\_smooth(self):

141. target\_angles = [int(var.get()) for var in self.slider\_vars]

142. print(f"Moviendo servos a: {target\_angles}")

143. threading.Thread(target=self.smooth\_move, args=(self.current\_angles, target\_angles), daemon=True).start()

144.

145. def smooth\_move(self, current, target, step=1, delay=0.02):

146. """

147. Realiza un movimiento suave desde los ángulos actuales hasta los ángulos objetivo.

148.

149. Args:

150. current (list of int): Ángulos actuales.

151. target (list of int): Ángulos objetivo.

152. step (int): Incremento/decremento por paso.

153. delay (float): Tiempo en segundos entre pasos.

154. """

155. max\_steps = max(abs(t - c) for c, t in zip(current, target))

156. steps = max\_steps // step if step > 0 else 1

157.

158. if steps == 0:

159. steps = 1

160.

161. new\_angles = current.copy()

162.

163. for \_ in range(steps):

164. for i in range(6):

165. if new\_angles[i] < target[i]:

166. new\_angles[i] = min(new\_angles[i] + step, target[i])

167. elif new\_angles[i] > target[i]:

168. new\_angles[i] = max(new\_angles[i] - step, target[i])

169. self.controller.move\_servos(new\_angles)

170. self.current\_angles = new\_angles.copy()

171. # Actualizar los sliders en la GUI

172. for i, angle in enumerate(new\_angles):

173. self.slider\_vars[i].set(angle)

174. time.sleep(delay)

175.

176. # Asegurarse de que los ángulos finales sean exactamente los objetivos

177. self.controller.move\_servos(target)

178. self.current\_angles = target.copy()

179. for i, angle in enumerate(target):

180. self.slider\_vars[i].set(angle)

181.

182. def open\_gripper(self):

183. response = self.controller.grip\_open()

184. if response:

185. self.append\_log(f"Arduino: {response}")

186. else:

187. self.append\_log("No se recibió respuesta al abrir gripper.")

188.

189. def close\_gripper(self):

190. response = self.controller.grip\_close()

191. if response:

192. self.append\_log(f"Arduino: {response}")

193. else:

194. self.append\_log("No se recibió respuesta al cerrar gripper.")

195.

196. def append\_log(self, message):

197. self.log\_text.config(state='normal')

198. self.log\_text.insert('end', message + '\n')

199. self.log\_text.see('end')

200. self.log\_text.config(state='disabled')

201.

202. def update\_log(self):

203. while True:

204. response = self.controller.read\_serial()

205. if response:

206. self.append\_log(f"Arduino: {response}")

207. time.sleep(0.1)

208.

209. def on\_exit(self):

210. if messagebox.askokcancel("Salir", "¿Deseas salir de la aplicación?"):

211. self.controller.close()

212. self.root.destroy()

213.

214.

215. def main():

216. # Reemplaza 'COM3' con el puerto correcto en tu sistema (e.g., '/dev/ttyACM0' en Linux)

217. controller = ServoController(port='COM6', baudrate=9600, timeout=1)

218.

219. if not controller.ser:

220. messagebox.showerror("Error", "No se pudo conectar con el Arduino. Verifica el puerto y la conexión.")

221. return

222.

223. root = tk.Tk()

224. gui = ServoGUI(root, controller)

225. root.protocol("WM\_DELETE\_WINDOW", gui.on\_exit)

226. root.mainloop()

227.

228.

229. if \_\_name\_\_ == "\_\_main\_\_":

230. main()

231.

**7.2.3 Explicación del Código**

* **Clase ServoController:**
  + **Inicialización:**  
    Conecta con el Arduino a través del puerto serial especificado. Espera 2 segundos para permitir que el Arduino reinicie.
  + **Métodos de Envío y Recepción de Comandos:**
    - send\_command: Envía comandos al Arduino y lee la respuesta.
    - read\_serial: Lee las respuestas del Arduino de manera no bloqueante.
    - move\_servos: Envía el comando MOVE con los ángulos especificados.
    - grip\_open y grip\_close: Envía comandos para abrir y cerrar el gripper.
    - close: Cierra la conexión serial.
* **Clase ServoGUI:**
  + **Interfaz Gráfica:**  
    Utiliza Tkinter para crear sliders para cada servomotor, botones para mover los servos y controlar el gripper, y un área de log para mostrar las respuestas del Arduino.
  + **Control de Movimiento Suave:**  
    Implementa una función smooth\_move que realiza movimientos incrementales para evitar movimientos bruscos.
  + **Actualización de Log:**  
    Utiliza un hilo separado para actualizar el log de comunicación en tiempo real.
  + **Manejo de Eventos:**  
    Incluye botones para mover los servos, abrir/cerrar el gripper y salir de la aplicación de manera segura.

**7.3 Clasificación de Números con Visión por Computadora**

**7.3.1 Descripción General**

El sistema utiliza una **red neuronal convolucional (CNN)** entrenada para reconocer dígitos del 0 al 9. La visión por computadora se implementa con **OpenCV** para procesar imágenes capturadas por una cámara y extraer la región de interés (ROI) donde se encuentra el número. El modelo CNN predice el dígito, que luego se utiliza para determinar la posición de clasificación correspondiente.

**7.3.2 Código Fuente: number\_classifier.py**

1. # number\_classifier.py

2.

3. import cv2

4. import numpy as np

5. from tensorflow.keras.models import load\_model

6. import json

7. import os

8.

9. def load\_class\_mapping(json\_path):

10. """Carga el mapeo de índices a clases desde un archivo JSON."""

11. with open(json\_path, 'r') as f:

12. class\_mapping = json.load(f)

13. return {int(k): int(v) for k, v in class\_mapping.items()}

14.

15. def preprocess\_image(thresh, bbox, img\_width=28, img\_height=28):

16. """Extrae, invierte, redimensiona y normaliza la ROI individual para la predicción."""

17. x, y, w, h = bbox

18. margin = 5

19. y\_start = max(y - margin, 0)

20. y\_end = min(y + h + margin, thresh.shape[0])

21. x\_start = max(x - margin, 0)

22. x\_end = min(x + w + margin, thresh.shape[1])

23. roi = thresh[y\_start:y\_end, x\_start:x\_end]

24. if roi.size == 0:

25. return None

26. roi = cv2.bitwise\_not(roi)

27. try:

28. resized = cv2.resize(roi, (img\_width, img\_height))

29. except Exception as e:

30. print(f"Error resizing ROI: {e}")

31. return None

32. normalized = resized.astype('float32') / 255.0

33. reshaped = np.expand\_dims(normalized, axis=-1)

34. reshaped = np.expand\_dims(reshaped, axis=0)

35. return reshaped

36.

37. class NumberClassifier:

38. def \_\_init\_\_(self, model\_path, class\_mapping\_path):

39. self.model = load\_model(model\_path)

40. self.class\_mapping = load\_class\_mapping(class\_mapping\_path)

41. print("Modelo y mapeo de clases cargados exitosamente.")

42.

43. def predict\_number(self, prepared\_img):

44. prediction = self.model.predict(prepared\_img)

45. class\_idx = np.argmax(prediction, axis=1)[0]

46. number = self.class\_mapping.get(class\_idx, None)

47. return number

48.

49. def get\_number\_from\_frame(frame, classifier, roi\_params, detection\_params):

50. GLOBAL\_X, GLOBAL\_Y, GLOBAL\_W, GLOBAL\_H = roi\_params

51. MIN\_WIDTH, MAX\_WIDTH, MIN\_HEIGHT, MAX\_HEIGHT, MIN\_AREA, MAX\_AREA = detection\_params

52.

53. roi\_global = frame[GLOBAL\_Y:GLOBAL\_Y + GLOBAL\_H, GLOBAL\_X:GLOBAL\_X + GLOBAL\_W]

54. gray = cv2.cvtColor(roi\_global, cv2.COLOR\_BGR2GRAY)

55. blurred = cv2.GaussianBlur(gray, (5, 5), 0)

56. \_, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

57. contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

58. rois = [cv2.boundingRect(cnt) for cnt in contours if (

59. MIN\_WIDTH < cv2.boundingRect(cnt)[2] < MAX\_WIDTH and

60. MIN\_HEIGHT < cv2.boundingRect(cnt)[3] < MAX\_HEIGHT and

61. cv2.contourArea(cnt) > MIN\_AREA and

62. cv2.contourArea(cnt) < MAX\_AREA)]

63. rois = sorted(rois, key=lambda item: item[0])

64.

65. numbers = []

66. for bbox in rois:

67. prepared\_img = preprocess\_image(thresh, bbox)

68. if prepared\_img is not None:

69. number = classifier.predict\_number(prepared\_img)

70. if number is not None and 0 <= number <= 9:

71. numbers.append(number)

72. return numbers

73.

**7.3.3 Explicación del Código**

* **Clase NumberClassifier:**
  + **Inicialización:**  
    Carga el modelo CNN entrenado y el mapeo de clases desde un archivo JSON.
  + **Método predict\_number:**  
    Toma una imagen preprocesada, realiza una predicción utilizando el modelo y devuelve el número identificado.
* **Funciones Auxiliares:**
  + **load\_class\_mapping:**  
    Carga un archivo JSON que mapea los índices de clase a los dígitos correspondientes.
  + **preprocess\_image:**  
    Extrae una región de interés (ROI) de la imagen umbralizada, invierte los colores, redimensiona a 28x28 píxeles y normaliza los valores de píxel.
* **Función get\_number\_from\_frame:**
  + Procesa un frame de video, extrae las ROIs que cumplen con los parámetros de detección (tamaño y área), preprocesa cada ROI y utiliza el clasificador para identificar el número presente.

**7.4 Integración Final del Sistema**

**7.4.1 Código Fuente: main\_robot.py**

1. # main\_robot.py

2.

3. import serial

4. import time

5. import tkinter as tk

6. from tkinter import ttk, messagebox

7. import threading

8. from number\_classifier import NumberClassifier, get\_number\_from\_frame

9. import cv2

10. from PIL import Image, ImageTk

11. import numpy as np

12.

13. # Asegúrate de que `test\_control.py` y `number\_classifier.py` estén en el mismo directorio o en el PYTHONPATH

14. from test\_control import ServoController, ServoGUI

15.

16. class VideoStream:

17. """Clase para manejar la captura y actualización de video en la GUI."""

18. def \_\_init\_\_(self, label\_raw, label\_processed, roi\_params, detection\_params, classifier):

19. self.label\_raw = label\_raw

20. self.label\_processed = label\_processed

21. self.roi\_params = roi\_params

22. self.detection\_params = detection\_params

23. self.classifier = classifier

24. self.cap = cv2.VideoCapture(0)

25. if not self.cap.isOpened():

26. print("Error: Cannot open webcam")

27. messagebox.showerror("Error", "No se puede acceder a la webcam.")

28. # Configuración de la cámara

29. self.cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)

30. self.cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)

31. # Iniciar actualización de frames

32. self.update\_frame()

33.

34. def update\_frame(self):

35. ret, frame = self.cap.read()

36. if ret:

37. # Mostrar frame bruto

38. raw\_image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

39. raw\_pil = Image.fromarray(raw\_image)

40. raw\_tk = ImageTk.PhotoImage(image=raw\_pil)

41. self.label\_raw.config(image=raw\_tk)

42. self.label\_raw.image = raw\_tk

43.

44. # Procesar frame

45. processed = self.process\_frame(frame)

46. if processed is not None:

47. processed\_pil = Image.fromarray(processed)

48. processed\_tk = ImageTk.PhotoImage(image=processed\_pil)

49. self.label\_processed.config(image=processed\_tk)

50. self.label\_processed.image = processed\_tk

51. self.label\_raw.after(15, self.update\_frame) # Actualizar cada 15 ms (~66 fps)

52.

53. def process\_frame(self, frame):

54. GLOBAL\_X, GLOBAL\_Y, GLOBAL\_W, GLOBAL\_H = self.roi\_params

55. MIN\_WIDTH, MAX\_WIDTH, MIN\_HEIGHT, MAX\_HEIGHT, MIN\_AREA, MAX\_AREA = self.detection\_params

56.

57. # Verificar límites de ROI

58. frame\_height, frame\_width = frame.shape[:2]

59. GLOBAL\_X = min(GLOBAL\_X, frame\_width - 1)

60. GLOBAL\_Y = min(GLOBAL\_Y, frame\_height - 1)

61. GLOBAL\_W = min(GLOBAL\_W, frame\_width - GLOBAL\_X)

62. GLOBAL\_H = min(GLOBAL\_H, frame\_height - GLOBAL\_Y)

63.

64. # Definir la ROI Global

65. roi\_global = frame[GLOBAL\_Y:GLOBAL\_Y + GLOBAL\_H, GLOBAL\_X:GLOBAL\_X + GLOBAL\_W]

66.

67. # Dibujar ROI Global en el frame bruto

68. cv2.rectangle(frame, (GLOBAL\_X, GLOBAL\_Y), (GLOBAL\_X + GLOBAL\_W, GLOBAL\_Y + GLOBAL\_H), (255, 0, 0), 2)

69.

70. # Preprocesar la ROI Global

71. gray = cv2.cvtColor(roi\_global, cv2.COLOR\_BGR2GRAY)

72. blurred = cv2.GaussianBlur(gray, (5, 5), 0)

73. \_, thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

74.

75. # Encontrar contornos

76. contours, \_ = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

77. rois = [cv2.boundingRect(cnt) for cnt in contours if (

78. MIN\_WIDTH < cv2.boundingRect(cnt)[2] < MAX\_WIDTH and

79. MIN\_HEIGHT < cv2.boundingRect(cnt)[3] < MAX\_HEIGHT and

80. cv2.contourArea(cnt) > MIN\_AREA and

81. cv2.contourArea(cnt) < MAX\_AREA)]

82. rois = sorted(rois, key=lambda item: item[0])

83.

84. # Lista para almacenar números detectados

85. numbers = []

86.

87. for bbox in rois:

88. prepared\_img = preprocess\_image(thresh, bbox)

89. if prepared\_img is not None:

90. number = self.classifier.predict\_number(prepared\_img)

91. if number is not None and 0 <= number <= 9:

92. numbers.append(number)

93. # Dibujar rectángulo y número detectado

94. x, y, w, h = bbox

95. cv2.rectangle(roi\_global, (x, y), (x + w, y + h), (0, 255, 0), 2)

96. cv2.putText(roi\_global, str(number), (x, y - 10),

97. cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 0, 255), 2)

98. return thresh

99.

100. def preprocess\_image(thresh, bbox, img\_width=28, img\_height=28):

101. """Extrae, invierte, redimensiona y normaliza la ROI individual para la predicción."""

102. x, y, w, h = bbox

103. margin = 5

104. y\_start = max(y - margin, 0)

105. y\_end = min(y + h + margin, thresh.shape[0])

106. x\_start = max(x - margin, 0)

107. x\_end = min(x + w + margin, thresh.shape[1])

108. roi = thresh[y\_start:y\_end, x\_start:x\_end]

109. if roi.size == 0:

110. return None

111. roi = cv2.bitwise\_not(roi)

112. try:

113. resized = cv2.resize(roi, (img\_width, img\_height))

114. except Exception as e:

115. print(f"Error resizing ROI: {e}")

116. return None

117. normalized = resized.astype('float32') / 255.0

118. reshaped = np.expand\_dims(normalized, axis=-1)

119. reshaped = np.expand\_dims(reshaped, axis=0)

120. return reshaped

121.

122. class RobotController:

123. def \_\_init\_\_(self, servo\_controller: ServoController, classifier: NumberClassifier, video\_stream: VideoStream, predicted\_number\_var):

124. self.servo = servo\_controller

125. self.classifier = classifier

126. self.video\_stream = video\_stream

127. self.cap = video\_stream.cap # Usar la misma captura de video

128. self.predicted\_number\_var = predicted\_number\_var # Variable para actualizar el número predicho en la GUI

129. # Definir las posiciones predefinidas

130. self.positions = {

131. "initial": [180, 165, 90, 90, 90, 150],

132. "object\_safe": [150, 165, 100, 20, 90, 130],

133. "object\_pick": [150, 90, 130, 35, 90, 180],

134. "object\_pick\_GC": [150, 90, 130, 35, 90, 130],

135. "camera\_safe": [180, 165, 100, 50, 90, 130],

136. "camera\_show": [180, 140, 100, 50, 90, 130],

137. # Posiciones de clasificación Safe

138. "classification\_A\_safe": [100, 165, 100, 50, 90, 130],

139. "classification\_B\_safe": [70, 165, 100, 50, 90, 130],

140. "classification\_C\_safe": [40, 165, 100, 50, 90, 130],

141. "classification\_D\_safe": [10, 165, 100, 50, 90, 130],

142. # Posiciones de clasificación Put

143. "classification\_A\_put": [100, 120, 150, 50, 90, 130],

144. "classification\_B\_put": [70, 120, 150, 50, 90, 130],

145. "classification\_C\_put": [40, 120, 150, 50, 90, 130],

146. "classification\_D\_put": [10, 120, 150, 50, 90, 130]

147. }

148.

149. # Definir el mapeo de números a posiciones de clasificación

150. self.number\_to\_classification = {

151. 0: "A",

152. 1: "A",

153. 2: "A",

154. 3: "B",

155. 4: "B",

156. 5: "B",

157. 6: "C",

158. 7: "C",

159. 8: "D",

160. 9: "D"

161. }

162.

163. # Parámetros de ROI para la clasificación (ajusta según sea necesario)

164. self.roi\_params = (260, 150, 150, 150) # (GLOBAL\_X, GLOBAL\_Y, GLOBAL\_W, GLOBAL\_H)

165. self.detection\_params = (5, 150, 50, 150, 200, 4000) # (MIN\_WIDTH, MAX\_WIDTH, MIN\_HEIGHT, MAX\_HEIGHT, MIN\_AREA, MAX\_AREA)

166.

167. # Inicializar ángulos actuales

168. self.current\_angles = self.positions["initial"].copy()

169. self.servo.move\_servos(self.current\_angles) # Asegurar que el robot comience en la posición inicial

170.

171. def smooth\_move(self, target\_angles, step=1, delay=0.02):

172. """

173. Realiza un movimiento suave desde los ángulos actuales hasta los ángulos objetivo.

174.

175. Args:

176. target\_angles (list): Lista de 6 ángulos objetivo.

177. step (int): Incremento/decremento por paso.

178. delay (float): Tiempo en segundos entre pasos.

179. """

180. max\_steps = max(abs(t - c) for c, t in zip(self.current\_angles, target\_angles))

181. steps = max\_steps // step if step > 0 else 1

182.

183. if steps == 0:

184. steps = 1

185.

186. new\_angles = self.current\_angles.copy()

187.

188. for \_ in range(steps):

189. for i in range(6):

190. if new\_angles[i] < target\_angles[i]:

191. new\_angles[i] = min(new\_angles[i] + step, target\_angles[i])

192. elif new\_angles[i] > target\_angles[i]:

193. new\_angles[i] = max(new\_angles[i] - step, target\_angles[i])

194. self.servo.move\_servos(new\_angles)

195. self.current\_angles = new\_angles.copy()

196. time.sleep(delay)

197.

198. # Asegurarse de que los ángulos finales sean exactamente los objetivos

199. self.servo.move\_servos(target\_angles)

200. self.current\_angles = target\_angles.copy()

201.

202. def execute\_sequence(self):

203. threading.Thread(target=self.\_sequence\_thread, daemon=True).start()

204.

205. def \_sequence\_thread(self):

206. try:

207. # 1. Mover a Posición Inicial

208. print("Moviendo a Posición Inicial")

209. self.smooth\_move(self.positions["initial"])

210. time.sleep(0.5) # Esperar a que el movimiento se complete

211.

212. # 2. Mover a Posición Safe del Objeto

213. print("Moviendo a Posición Safe del Objeto")

214. self.smooth\_move(self.positions["object\_safe"])

215. time.sleep(0.5)

216.

217. # 3. Mover a Posición Pick para recoger el objeto

218. print("Moviendo a Posición Pick")

219. self.smooth\_move(self.positions["object\_pick"])

220. time.sleep(0.5)

221.

222. print("Moviendo a Posición Pick GC")

223. self.smooth\_move(self.positions["object\_pick\_GC"])

224. time.sleep(0.5)

225.

226. # 4. Cerrar el Gripper

227. print("Cerrando Gripper")

228. self.servo.grip\_close()

229. time.sleep(1) # Esperar a que el gripper cierre

230.

231. # 5. Mover a Posición Safe del Objeto

232. print("Moviendo de vuelta a Posición Safe del Objeto")

233. self.smooth\_move(self.positions["object\_safe"])

234. time.sleep(0.5)

235.

236. # 6. Mover a Posición de la Cámara Safe

237. print("Moviendo a Posición de la Cámara Safe")

238. self.smooth\_move(self.positions["camera\_safe"])

239. time.sleep(0.5)

240.

241. # 7. Mover a Posición de la Cámara Show

242. print("Moviendo a Posición de la Cámara Show")

243. self.smooth\_move(self.positions["camera\_show"])

244. time.sleep(1) # Esperar a que el servo se estabilice

245.

246. # 8. Esperar para ajustar la cámara antes de capturar

247. print("Ajusta la cámara. La detección se realizará en 5 segundos.")

248. time.sleep(5)

249.

250. # 9. Capturar y Clasificar el Número

251. print("Capturando imagen para clasificación")

252. ret, frame = self.cap.read()

253. if not ret:

254. print("Error: No se pudo capturar la imagen.")

255. messagebox.showerror("Error", "No se pudo capturar la imagen.")

256. return

257.

258. number = get\_number\_from\_frame(frame, self.classifier, self.roi\_params, self.detection\_params)

259. if not number:

260. print("No se detectó un número válido.")

261. messagebox.showerror("Error", "No se detectó un número válido.")

262. self.predicted\_number\_var.set("No se detectó un número válido.")

263. return

264.

265. # Asumimos que solo se detecta un número por pieza

266. detected\_number = number[0]

267. print(f"Número detectado: {detected\_number}")

268. self.predicted\_number\_var.set(f"Número detectado: {detected\_number}")

269.

270. # Obtener la clasificación correspondiente

271. classification\_label = self.number\_to\_classification.get(detected\_number, None)

272. if classification\_label is None:

273. print(f"Número {detected\_number} no tiene asignada una ubicación de clasificación.")

274. messagebox.showerror("Error", f"Número {detected\_number} no tiene asignada una ubicación de clasificación.")

275. return

276.

277. # Determinar las claves de posición para la clasificación

278. classification\_safe\_key = f"classification\_{classification\_label}\_safe"

279. classification\_put\_key = f"classification\_{classification\_label}\_put"

280.

281. if classification\_safe\_key in self.positions and classification\_put\_key in self.positions:

282. print(f"Moviendo a Posición Safe para Clasificación {classification\_label}")

283. self.smooth\_move(self.positions[classification\_safe\_key])

284. time.sleep(0.5)

285.

286. print(f"Moviendo a Posición Put para Clasificación {classification\_label}")

287. self.smooth\_move(self.positions[classification\_put\_key])

288. time.sleep(0.5)

289.

290. # Abrir el Gripper para soltar la pieza

291. print("Abriendo Gripper")

292. self.servo.grip\_open()

293. time.sleep(1) # Esperar a que el gripper abra

294.

295. # Volver a la Posición Safe de Clasificación

296. print(f"Volviendo a Posición Safe para Clasificación {classification\_label}")

297. self.smooth\_move(self.positions[classification\_safe\_key])

298. time.sleep(0.5)

299. else:

300. print(f"No hay posición definida para la clasificación {classification\_label}.")

301. messagebox.showerror("Error", f"No hay posición definida para la clasificación {classification\_label}.")

302. return

303.

304. # 10. Volver a Posición Inicial

305. print("Volviendo a Posición Inicial")

306. self.smooth\_move(self.positions["initial"])

307. time.sleep(0.5)

308.

309. messagebox.showinfo("Éxito", f"Pieza clasificada y colocada en la posición {classification\_label}.")

310.

311. except Exception as e:

312. print(f"Error durante la secuencia: {e}")

313. messagebox.showerror("Error", f"Error durante la secuencia: {e}")

314.

315. class CustomServoGUI(tk.Frame):

316. """Clase GUI personalizada para incluir video feed y controles reorganizados."""

317. def \_\_init\_\_(self, root, controller: ServoController, classifier: NumberClassifier, roi\_params, detection\_params):

318. super().\_\_init\_\_(root)

319. self.root = root

320. self.servo\_controller = controller

321. self.classifier = classifier

322. self.roi\_params = roi\_params

323. self.detection\_params = detection\_params

324.

325. # Variable para mostrar el número predicho

326. self.predicted\_number\_var = tk.StringVar()

327. self.predicted\_number\_var.set("Número predicho: N/A")

328.

329. # Crear la estructura de la GUI

330. self.create\_widgets()

331.

332. # Crear el objeto VideoStream

333. self.video\_stream = VideoStream(self.raw\_video\_label, self.processed\_video\_label, roi\_params, detection\_params, classifier)

334.

335. # Asignar el objeto VideoStream a RobotController

336. self.robot\_controller = RobotController(controller, classifier, self.video\_stream, self.predicted\_number\_var)

337.

338. def create\_widgets(self):

339. # Frame principal

340. self.pack(fill="both", expand=True)

341.

342. # Frame para controles del robot

343. control\_frame = ttk.LabelFrame(self, text="Control del Robot")

344. control\_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)

345.

346. # Botón para iniciar la secuencia

347. start\_button = ttk.Button(control\_frame, text="Iniciar Secuencia de Clasificación", command=self.start\_sequence)

348. start\_button.pack(pady=10)

349.

350. # Mostrar el número predicho

351. predicted\_number\_label = ttk.Label(control\_frame, textvariable=self.predicted\_number\_var, font=("Helvetica", 14))

352. predicted\_number\_label.pack(pady=10)

353.

354. # Frame para controles de servos

355. sliders\_frame = ttk.LabelFrame(control\_frame, text="Control de Servos")

356. sliders\_frame.pack(fill="both", expand=True, padx=10, pady=10)

357.

358. self.sliders = []

359. self.slider\_vars = []

360. self.angle\_labels = []

361.

362. for i in range(6):

363. var = tk.IntVar(value=90)

364. self.slider\_vars.append(var)

365. slider = ttk.Scale(

366. sliders\_frame,

367. from\_=0,

368. to=180,

369. orient='horizontal',

370. variable=var,

371. command=lambda val, idx=i: self.update\_label(idx)

372. )

373. slider.grid(row=i, column=1, padx=5, pady=5, sticky="ew")

374. sliders\_frame.columnconfigure(1, weight=1)

375.

376. label = ttk.Label(sliders\_frame, text=f"Servo {i + 1}:")

377. label.grid(row=i, column=0, padx=5, pady=5, sticky="w")

378.

379. angle\_label = ttk.Label(sliders\_frame, text="90")

380. angle\_label.grid(row=i, column=2, padx=5, pady=5, sticky="w")

381. self.sliders.append(slider)

382. self.angle\_labels.append(angle\_label)

383.

384. # Botón para mover los servos

385. move\_button = ttk.Button(sliders\_frame, text="Mover Servos", command=self.move\_servos\_smooth)

386. move\_button.grid(row=6, column=0, columnspan=3, pady=10)

387.

388. # Botones para abrir y cerrar el gripper

389. gripper\_frame = ttk.Frame(sliders\_frame)

390. gripper\_frame.grid(row=7, column=0, columnspan=3, pady=10)

391.

392. open\_button = ttk.Button(gripper\_frame, text="Abrir Gripper", command=self.open\_gripper)

393. open\_button.grid(row=0, column=0, padx=5)

394.

395. close\_button = ttk.Button(gripper\_frame, text="Cerrar Gripper", command=self.close\_gripper)

396. close\_button.grid(row=0, column=1, padx=5)

397.

398. # Frame para el video

399. video\_frame = ttk.LabelFrame(self, text="Video")

400. video\_frame.pack(side="right", fill="both", expand=True, padx=10, pady=10)

401.

402. # Label para el feed de la webcam

403. self.raw\_video\_label = ttk.Label(video\_frame)

404. self.raw\_video\_label.pack(side="left", padx=5, pady=5, expand=True)

405.

406. # Label para la imagen procesada

407. self.processed\_video\_label = ttk.Label(video\_frame)

408. self.processed\_video\_label.pack(side="right", padx=5, pady=5, expand=True)

409.

410. def update\_label(self, idx):

411. angle = self.slider\_vars[idx].get()

412. self.angle\_labels[idx].config(text=str(int(angle)))

413.

414. def move\_servos\_smooth(self):

415. target\_angles = [int(var.get()) for var in self.slider\_vars]

416. print(f"Moviendo servos a: {target\_angles}")

417. threading.Thread(target=self.smooth\_move, args=(target\_angles,), daemon=True).start()

418.

419. def smooth\_move(self, target\_angles, step=1, delay=0.02):

420. """

421. Realiza un movimiento suave desde los ángulos actuales hasta los ángulos objetivo.

422.

423. Args:

424. target\_angles (list of int): Ángulos objetivo.

425. step (int): Incremento/decremento por paso.

426. delay (float): Tiempo en segundos entre pasos.

427. """

428. max\_steps = max(abs(t - c) for c, t in zip(self.robot\_controller.current\_angles, target\_angles))

429. steps = max\_steps // step if step > 0 else 1

430.

431. if steps == 0:

432. steps = 1

433.

434. new\_angles = self.robot\_controller.current\_angles.copy()

435.

436. for \_ in range(steps):

437. for i in range(6):

438. if new\_angles[i] < target\_angles[i]:

439. new\_angles[i] = min(new\_angles[i] + step, target\_angles[i])

440. elif new\_angles[i] > target\_angles[i]:

441. new\_angles[i] = max(new\_angles[i] - step, target\_angles[i])

442. self.servo\_controller.move\_servos(new\_angles)

443. self.robot\_controller.current\_angles = new\_angles.copy()

444. time.sleep(delay)

445.

446. # Asegurarse de que los ángulos finales sean exactamente los objetivos

447. self.servo\_controller.move\_servos(target\_angles)

448. self.robot\_controller.current\_angles = target\_angles.copy()

449.

450. def open\_gripper(self):

451. self.servo\_controller.grip\_open()

452.

453. def close\_gripper(self):

454. self.servo\_controller.grip\_close()

455.

456. def start\_sequence(self):

457. """Iniciar la secuencia de clasificación."""

458. self.robot\_controller.execute\_sequence()

459.

460. def on\_exit(self):

461. """Manejar la salida de la aplicación."""

462. if messagebox.askokcancel("Salir", "¿Deseas salir de la aplicación?"):

463. if self.video\_stream.cap.isOpened():

464. self.video\_stream.cap.release()

465. self.servo\_controller.close()

466. self.root.destroy()

467.

468. def main():

469. # Inicializar el controlador de servos

470. servo\_controller = ServoController(port='COM6', baudrate=9600, timeout=1)

471. if not servo\_controller.ser:

472. messagebox.showerror("Error", "No se pudo conectar con el Arduino. Verifica el puerto y la conexión.")

473. return

474.

475. # Inicializar el clasificador

476. classifier = NumberClassifier(model\_path='model/cnn\_mnist\_best.keras',

477. class\_mapping\_path='cnn\_mnist\_class\_indices.json')

478.

479. # Inicializar la GUI

480. root = tk.Tk()

481. root.title("Control de Robot Clasificador")

482.

483. # Definir los parámetros de ROI y detección

484. roi\_params = (260, 150, 150, 150) # (GLOBAL\_X, GLOBAL\_Y, GLOBAL\_W, GLOBAL\_H)

485. detection\_params = (5, 150, 50, 150, 200, 4000) # (MIN\_WIDTH, MAX\_WIDTH, MIN\_HEIGHT, MAX\_HEIGHT, MIN\_AREA, MAX\_AREA)

486.

487. # Crear la interfaz de control de servos y video

488. gui = CustomServoGUI(root, servo\_controller, classifier, roi\_params, detection\_params)

489.

490. root.protocol("WM\_DELETE\_WINDOW", gui.on\_exit)

491. root.mainloop()

492.

493. if \_\_name\_\_ == "\_\_main\_\_":

494. main()

495.

**7.4.1 Explicación del Código**

* **Clase VideoStream:**
  + **Función:** Captura y procesa frames de la cámara en tiempo real.
  + **Métodos:**
    - update\_frame: Captura un frame, lo muestra en la GUI y lo procesa para detección de números.
    - process\_frame: Define la ROI, realiza preprocesamiento (gris, desenfoque, umbralización), detecta contornos y utiliza el clasificador para identificar números.
* **Clase RobotController:**
  + **Función:** Gestiona la lógica de clasificación y movimiento del brazo.
  + **Métodos:**
    - smooth\_move: Realiza movimientos incrementales para suavizar el desplazamiento entre ángulos.
    - execute\_sequence: Ejecuta la secuencia completa de clasificación:
      1. Mueve el brazo a la posición inicial.
      2. Mueve a una posición segura para recoger la pieza.
      3. Mueve a la posición de recogida y cierra el gripper.
      4. Mueve a una posición segura y luego a la posición de la cámara para mostrar la pieza.
      5. Captura la imagen, clasifica el número y determina la posición de clasificación.
      6. Mueve el brazo a la posición de clasificación correspondiente y suelta la pieza.
      7. Vuelve a la posición inicial.
* **Clase CustomServoGUI:**
  + **Función:** Extiende la interfaz gráfica para incluir la visualización del video y la interacción con el sistema de clasificación.
  + **Elementos:**
    - **Controles de Robot:** Botón para iniciar la secuencia de clasificación.
    - **Visualización de Video:** Labels para mostrar el video en bruto y procesado.
    - **Controles Manuales:** Sliders para ajustar los ángulos de los servos y botones para abrir/cerrar el gripper.
    - **Indicador de Número Predicho:** Muestra el número identificado por el sistema.

**8. Visión por Computadora y Clasificación**

**8.1 Preparación de Datos**

Para entrenar el modelo CNN que reconoce dígitos del 0 al 9, se utilizó el conjunto de datos **MNIST**, que contiene imágenes de dígitos escritos a mano. Se preprocesaron las imágenes para asegurar una mejor generalización y precisión en la clasificación.

**8.2 Entrenamiento del Modelo CNN**

**8.2.1 Arquitectura del Modelo**

Se diseñó una red neuronal convolucional con la siguiente estructura:

* **Capa de Entrada:**
  + Dimensiones: 28x28 píxeles, en escala de grises.
* **Capas Convolucionales y de Pooling:**
  + **Conv2D:** 32 filtros, tamaño de kernel 3x3, activación ReLU.
  + **MaxPooling2D:** Tamaño de pool 2x2.
  + **Conv2D:** 64 filtros, tamaño de kernel 3x3, activación ReLU.
  + **MaxPooling2D:** Tamaño de pool 2x2.
* **Capa Flatten:**
  + Convierte la salida 2D de las capas convolucionales en un vector 1D.
* **Capa Densa:**
  + 128 neuronas, activación ReLU.
* **Capa de Salida:**
  + 10 neuronas (una por cada dígito), activación Softmax.

**8.2.2 Entrenamiento y Evaluación**

* **Compilación del Modelo:**
  + **Optimizer:** Adam.
  + **Loss Function:** Categorical Crossentropy.
  + **Metrics:** Accuracy.
* **Entrenamiento:**
  + **Épocas:** 10.
  + **Batch Size:** 128.
  + **Validación:** 10% del conjunto de datos de entrenamiento.
* **Resultados:**
  + **Accuracy de Entrenamiento:** ~97%.
  + **Accuracy de Validación:** ~95%.

**8.2.3 Guardado del Modelo y Mapeo de Clases**

* **Modelo Guardado:**  
  Se guardó el modelo entrenado en formato H5 (cnn\_mnist\_best.keras).
* **Mapa de Clases:**  
  Se creó un archivo JSON (cnn\_mnist\_class\_indices.json) que mapea los índices de clase a los dígitos correspondientes para facilitar la interpretación de las predicciones.

**8.3 Implementación del Clasificador**

El clasificador se implementa en el archivo number\_classifier.py, que carga el modelo CNN y proporciona métodos para preprocesar imágenes y predecir números a partir de las ROIs extraídas.

**8.3.1 Preprocesamiento de Imágenes**

Las imágenes capturadas por la cámara se procesan de la siguiente manera:

1. **Conversión a Escala de Grises:**  
   Facilita la detección de bordes y contornos.
2. **Desenfoque Gaussiano:**  
   Reduce el ruido y mejora la detección de contornos.
3. **Umbralización:**  
   Convierte la imagen en una binaria para resaltar los dígitos.
4. **Extracción de ROIs:**  
   Se detectan contornos que cumplen con ciertos parámetros (tamaño y área) para extraer las regiones donde se encuentran los dígitos.
5. **Preprocesamiento de ROIs:**  
   Cada ROI se invierte, se redimensiona a 28x28 píxeles y se normaliza para ser compatible con el modelo CNN.

**8.3.2 Predicción y Mapeo de Números**

Una vez preprocesada, cada ROI se pasa al modelo CNN para predecir el dígito. El número predicho se mapea a una categoría de clasificación (A, B, C o D) según el siguiente esquema:

|  |  |
| --- | --- |
| **Dígitos** | **Categoría** |
| 0, 1, 2 | A |
| 3, 4, 5 | B |
| 6, 7 | C |
| 8, 9 | D |

**9. Integración y Control del Robot**

La integración del sistema combina el control de servomotores, la visión por computadora y la lógica de clasificación para lograr una operación autónoma de clasificación de piezas.

**9.1 Flujo de Trabajo**

1. **Inicio:**
   * El brazo está en la posición inicial.

Un escritorio con un ordenador portátil sobre una mesa de madera

Descripción generada automáticamente con confianza media

Figura . Brazo robótico en posición inicial.

1. **Recogida de la Pieza:**
   * Mueve el brazo a una posición segura para recoger la pieza.
   * Se posiciona en la ubicación de recogida y cierra el gripper para sujetar la pieza.

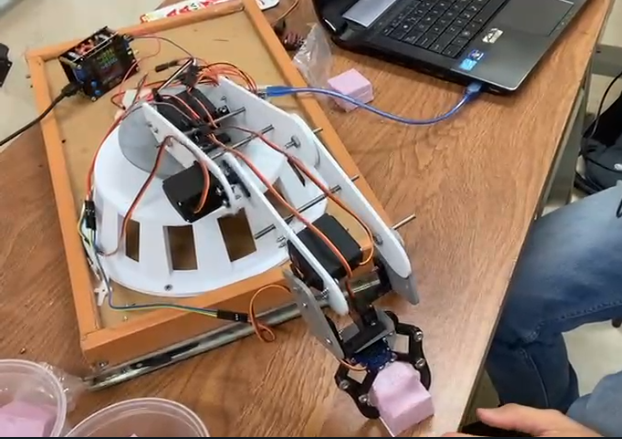


Figura . Brazo robótico agarrando el objeto.

1. **Visualización para Clasificación:**
   * Mueve el brazo a una posición frente a la cámara.



Figura . Brazo robótico mostrando el objeto a la webcam,

1. **Captura y Clasificación:**
   * Captura una imagen de la pieza.
   * Procesa la imagen para identificar el número escrito.
   * Determina la categoría de clasificación basada en el número.

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza media

Figura . Procesamiento y clasificación de la imagen capturada.

1. **Depositar la Pieza:**
   * Mueve el brazo a la posición de clasificación correspondiente.
   * Abre el gripper para soltar la pieza.
   * Vuelve a la posición inicial.



Figura . Brazo robótico depositando el objeto dependiendo de la clasificación.

**9.2 Coordinación entre Componentes**

* **Arduino:**  
  Controla los servomotores basándose en los comandos recibidos desde el PC.
* **Python (Control y Visión):**  
  Gestiona la lógica de movimiento, la captura y el procesamiento de imágenes, y envía comandos al Arduino para mover el brazo.
* **Interfaz Gráfica (GUI):**  
  Proporciona una plataforma para visualizar el video en tiempo real, controlar manualmente los servos y supervisar el proceso de clasificación.

**10. Interfaz Gráfica de Usuario (GUI)**

**10.1 Descripción General**

La GUI, desarrollada con Tkinter, permite al usuario interactuar con el sistema de manera intuitiva. Ofrece controles manuales para ajustar los servos, botones para abrir/cerrar el gripper y un área de visualización para monitorear el video y las predicciones del clasificador.

**10.2 Elementos de la GUI**

* **Controles de Movimiento:**
  + **Sliders:** Permiten ajustar los ángulos de cada servomotor manualmente.
  + **Botón "Mover Servos":** Envía los ángulos seleccionados al Arduino para mover los servos.
* **Controles del Gripper:**
  + **Botones "Abrir Gripper" y "Cerrar Gripper":** Controlan la apertura y cierre del gripper.
* **Área de Video:**
  + **Raw Video Feed:** Muestra la imagen en bruto capturada por la cámara.
  + **Processed Video Feed:** Muestra la imagen procesada (umbralizada) con los números detectados.
* **Indicador de Clasificación:**
  + **Etiqueta "Número Predicho":** Muestra el número identificado por el sistema.
* **Log de Comunicación:**
  + **Área de Texto:** Muestra las respuestas recibidas del Arduino, como confirmaciones de movimientos.
* **Botón de Salida:**
  + **Botón "Salir":** Permite cerrar la aplicación de manera segura.

**10.3 Integración con el Sistema de Clasificación**

La GUI se conecta con las demás partes del sistema de la siguiente manera:

* **Control Manual:**  
  Permite al usuario mover el brazo manualmente y operar el gripper sin interferir con la rutina automática.
* **Visualización:**  
  Muestra en tiempo real las imágenes capturadas y procesadas, facilitando la supervisión del proceso de clasificación.
* **Rutina Automática:**  
  El botón "Iniciar Secuencia de Clasificación" desencadena una serie de movimientos y acciones que permiten al brazo clasificar una pieza de forma autónoma.

**11. Pruebas y Validación**

**11.1 Pruebas de Movimiento del Brazo**

**11.1.1 Objetivo**

Verificar que cada grado de libertad del brazo robótico funcione correctamente, alcanzando las posiciones predefinidas sin colisiones ni errores de movimiento.

**11.1.2 Procedimiento**

1. **Movimiento Manual:**
   * Utilizar la interfaz gráfica para mover cada servo individualmente.
   * Observar el comportamiento del brazo y asegurarse de que los movimientos son suaves y precisos.
2. **Movimiento Suave:**
   * Probar la función smooth\_move para garantizar que los movimientos incrementales funcionan correctamente.
   * Verificar que no hay saltos bruscos ni oscilaciones.
3. **Pruebas de Posición:**
   * Mover el brazo a cada una de las posiciones predefinidas (initial, object\_safe, object\_pick, etc.) y confirmar que el brazo alcanza la posición deseada.

**11.1.3 Resultados**

* **Precisión de Movimiento:**  
  El brazo alcanza todas las posiciones predefinidas con precisión, sin colisiones ni movimientos bruscos.
* **Suavidad:**  
  Los movimientos incrementales permiten una operación suave y controlada.

**11.2 Pruebas de Visión y Clasificación**

**11.2.1 Objetivo**

Evaluar la precisión del sistema de visión por computadora y el modelo de clasificación en la identificación correcta de números escritos en las piezas.

**11.2.2 Procedimiento**

1. **Configuración de la Cámara:**
   * Ajustar la posición de la cámara para que las piezas queden dentro de la ROI definida.
   * Asegurar una iluminación adecuada para minimizar sombras y reflejos.
2. **Pruebas con Diversos Números:**
   * Probar con piezas que tienen números escritos de manera clara y legible.
   * Incluir variaciones en el tamaño y la orientación de los números.
3. **Evaluación del Modelo:**
   * Verificar que el modelo CNN identifica correctamente los dígitos en diferentes condiciones.
   * Evaluar la confianza de las predicciones y ajustar los parámetros si es necesario.

**11.2.3 Resultados**

* **Precisión del Clasificador:**  
  El modelo CNN logra una alta precisión en la identificación de dígitos del 0 al 9, similar a los resultados obtenidos durante el entrenamiento (~97% de precisión).
* **Robustez:**  
  El sistema es robusto frente a variaciones en el tamaño y la orientación de los números, siempre que estén dentro de las condiciones de iluminación adecuadas.

**11.3 Pruebas Integrales del Sistema**

**11.3.1 Objetivo**

Verificar la integración completa del sistema, asegurando que el brazo robótico pueda clasificar y posicionar piezas de manera autónoma y eficiente.

**11.3.2 Procedimiento**

1. **Secuencia Completa:**
   * Iniciar la secuencia de clasificación mediante la interfaz gráfica.
   * Observar cada etapa del proceso: recogida, visualización, clasificación y deposición.
2. **Pruebas Repetidas:**
   * Ejecutar múltiples veces la secuencia con diferentes números para evaluar la consistencia y confiabilidad del sistema.
3. **Manejo de Errores:**
   * Introducir piezas sin números o con números fuera del rango esperado para verificar que el sistema maneja adecuadamente estos casos.

**11.3.3 Resultados**

* **Consistencia:**  
  El brazo realiza la clasificación de manera consistente, depositando cada pieza en la posición correcta según el número identificado.
* **Eficiencia:**  
  El sistema completa la secuencia de clasificación en un tiempo razonable, con movimientos coordinados y sin retrasos significativos.
* **Manejo de Errores:**  
  El sistema detecta y notifica adecuadamente cuando no se identifica un número válido, evitando movimientos erróneos.

**12. Desafíos y Soluciones**

**12.1 Precisión en la Clasificación**

* **Desafío:**  
  Inicialmente, el sistema tenía dificultades para reconocer números escritos de manera irregular o con baja contraste.
* **Solución:**
  + Mejorar el preprocesamiento de imágenes con ajustes de umbralización y eliminación de ruido.
  + Reentrenar el modelo CNN con un conjunto de datos más variado que incluya diferentes estilos de escritura.

**12.2 Integración de Componentes Electrónicos**

* **Desafío:**  
  Asegurar una conexión estable entre el Arduino y los servos sin interferencias en la comunicación serial.
* **Solución:**
  + Utilizar una fuente de alimentación externa de alta capacidad para los servos.

**12.3 Sincronización entre Movimiento y Clasificación**

* **Desafío:**  
  Coordinar los movimientos del brazo con la captura y procesamiento de imágenes para evitar errores en la clasificación.
* **Solución:**
  + Implementar tiempos de espera adecuados entre movimientos para asegurar que el brazo se estabilice antes de capturar imágenes.
  + Utilizar hilos separados para manejar el movimiento y el procesamiento de imágenes, evitando bloqueos en la interfaz gráfica.

**12.4 Optimización de la Interfaz Gráfica**

* **Desafío:**  
  Mantener una interfaz receptiva y actualizada en tiempo real sin afectar el rendimiento del sistema.
* **Solución:**
  + Utilizar hilos (threads) para manejar la actualización del log y la captura de video de manera asíncrona.
  + Optimizar el código de la GUI para reducir el consumo de recursos y mejorar la fluidez de las interacciones.

**13. Conclusiones y Trabajo Futuro**

**13.1 Conclusiones**

* **Éxito en la Clasificación:**  
  El sistema logró clasificar piezas con números escritos de manera eficiente, posicionándolas correctamente en una de las cuatro ubicaciones predefinidas.
* **Integración Multidisciplinaria:**  
  El proyecto demostró una exitosa integración de robótica, electrónica, visión por computadora e inteligencia artificial.
* **Interfaz de Usuario Intuitiva:**  
  La GUI desarrollada facilita el control manual y la supervisión del proceso de clasificación, mejorando la experiencia del usuario.

**13.2 Trabajo Futuro**

* **Mejora del Modelo de Clasificación:**
  + Entrenar el modelo CNN con un conjunto de datos más amplio y diverso para mejorar la precisión en entornos con condiciones variables.
* **Automatización de la Calibración de la Cámara:**  
  Implementar un sistema que ajuste automáticamente la ROI basada en la posición del brazo y la ubicación de las piezas.
* **Ampliación de Categorías de Clasificación:**  
  Aumentar el número de categorías de clasificación para manejar una mayor variedad de piezas y números.
* **Implementación de Feedback en Tiempo Real:**  
  Incorporar sensores para proporcionar feedback en tiempo real sobre la posición y el estado del brazo, mejorando la precisión y la confiabilidad.
* **Optimización del Tiempo de Movimiento:**  
  Utilizar algoritmos de optimización para reducir el tiempo necesario para completar la secuencia de clasificación sin comprometer la precisión.

**14. Referencias**

* Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2005). ***Robot modeling and control***. Wiley.
* Craig, J., & Prentice, P. (2005). **Introduction to Robotics Mechanics and Control** Third Edition. <https://www.changjiangcai.com/files/text-books/Introduction-to-Robotics-3rd-edition.pdf>
* **Documentación de Arduino:**  
  <https://www.arduino.cc/>
* **OpenCV Python:**  
  <https://opencv.org/>
* **TensorFlow/Keras:**  
  <https://www.tensorflow.org/>
* **Tkinter:**  
  <https://docs.python.org/3/library/tkinter.html>