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.

Figura 1. 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.

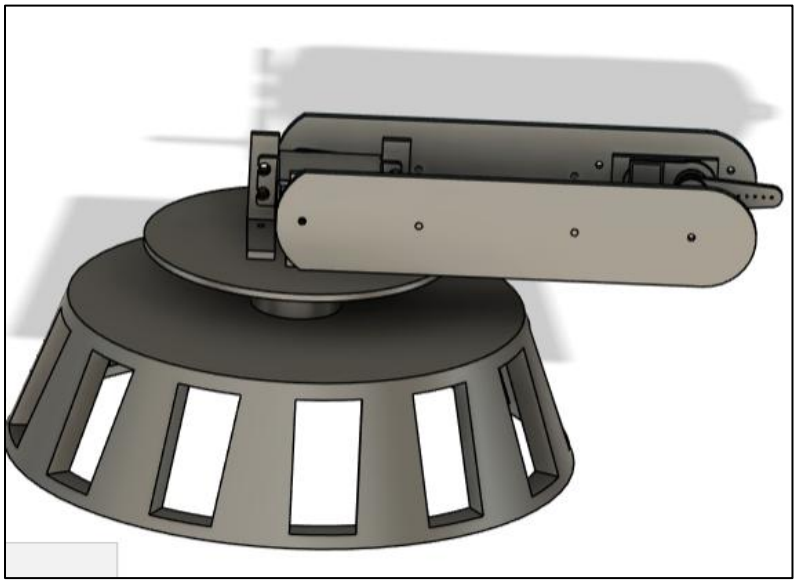

Figura 2. Unión del Plato con el Eslabón 1.

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.

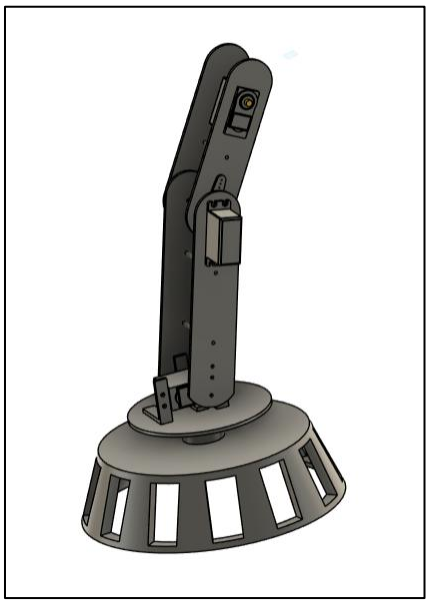

Figura 3. Unión del Plato y Eslabón 1 con el Eslabón 2.

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.

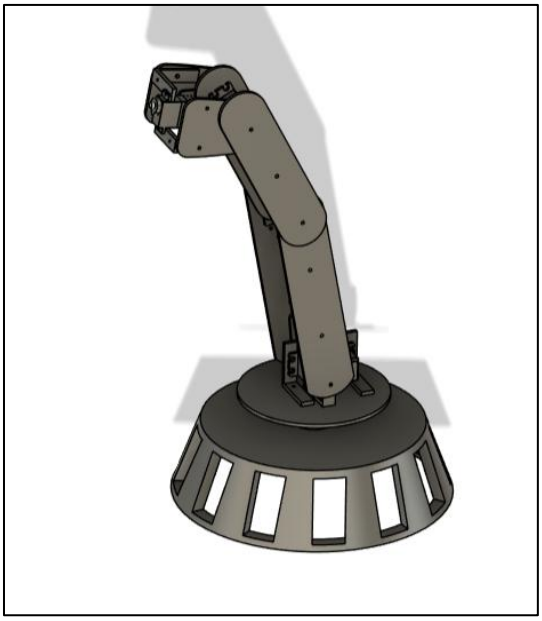

Figura 4. Unión del Plato, Eslabón 1 y Eslabón 2 con el Eslabón 3.

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.

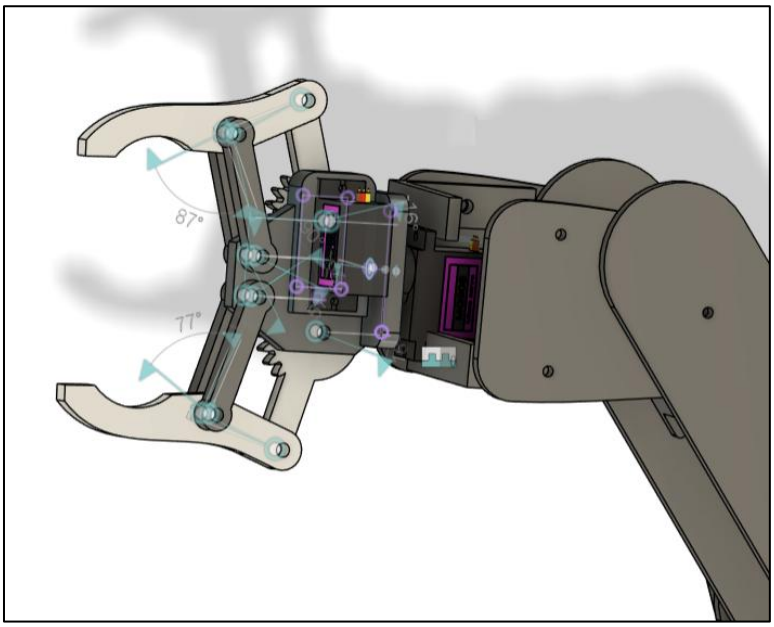

Figura 5. Unión del Gripper con el Eslabón 3.

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.

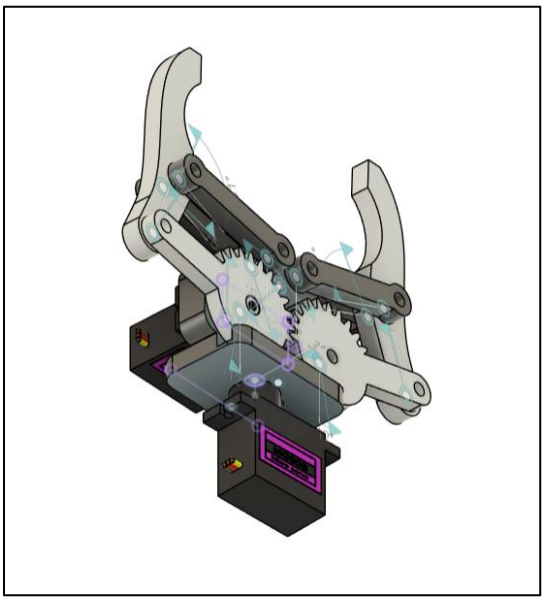

Figura 6. Mecanismo del Gripper.

---

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 7. 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 A_i

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

Eslabón	$\boldsymbol{\theta_I}$	$\boldsymbol{d_i(cm)}$	$\boldsymbol{a_i(cm)}$	$\boldsymbol{\alpha_i(°)}$
1	θ_1	12	0	+90°
2	θ_2	0	15	0°
3	θ_3	0	10	0°
4	θ_4	0	0	-90°

5	θ_5	18.1	0	0°

La matriz D-H genérica es:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz A_1

Parámetros: $\theta_1 = T_1, d_1 = 12, a_1 = 0, \alpha_1 = 90°$.

$Para\ i = 1 \rightarrow\ \cos\alpha_1 = \cos(90) = 0,\ \sin\alpha_1 = \sin 90 = 1$

$$A_1 = \begin{bmatrix} \cos T_1 & -\sin T_1(0) & \sin T_1(1) & 0 \\ \sin T_1 & \cos T_1(0) & -\cos T_1(1) & 0 \\ 0 & 1 & 0 & 12 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos T_1 & 0 & \sin T_1 & 0 \\ \sin T_1 & 0 & -\cos T_1 & 0 \\ 0 & 1 & 0 & 12 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz A_2

Parámetros: $\theta_2 = T_2, d_2 = 0, a_2 = 15, \alpha_2 = 0°$.

$Para\ i = 2 \rightarrow\ \cos\alpha_2 = 1, \sin\alpha_2 = 0$

$$A_2 = \begin{bmatrix} \cos T_2 & -\sin T_2 & 0 & 15\cos T_2 \\ \sin T_2 & \cos T_2 & 0 & 15\sin T_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz A_3

Parámetros: $\theta_3 = T_3, d_3 = 0, a_3 = 10, \alpha_3 = 0°$.

$Para\ i = 3 \rightarrow\ \cos\alpha_3 = 1, \sin\alpha_3 = 0$

$$A_3 = \begin{bmatrix} \cos T_3 & -\sin T_3 & 0 & 10\cos T_3 \\ \sin T_3 & \cos T_3 & 0 & 10\sin T_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz A_4

Parámetros: $\theta_4 = T_4, d_4 = 0, a_4 = 10, \alpha_4 = -90°$.

$Para\ i = 4 \rightarrow\ \cos\alpha_4 = 0, \sin\alpha_4 = -1$

$$A_4 = \begin{bmatrix} \cos T_4 & -\sin T_4(0) & \sin T_4(-1) & 0 \\ \sin T_4 & \cos T_4(0) & -\cos T_4(-1) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos T_4 & 0 & -\sin T_4 & 0 \\ \sin T_4 & 0 & \cos T_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz A_5

Parámetros: $\theta_5 = T_5$, $d_5 = 18.1\ cm$, $a_5 = 0$, $\alpha_5 = 0°$.

$Para\ i = 5 \rightarrow\ \cos\alpha_5 = 1, \sin\alpha_5 = 0$

$$A_5 = \begin{bmatrix} \cos T_5 & -\sin T_5 & 0 & 0 \\ \sin T_5 & \cos T_5 & 0 & 0 \\ 0 & 0 & 1 & 18.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz de Transformación Final

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

$$^0T_5 = A_1A_2A_3A_4A_5$$

$$T_5^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se obtiene los siguientes coeficientes:

$$R_{11} = \cos(T5)\big(\cos(T4)\big(\cos(T1)\cos(T2)\cos(T3) - \cos(T1)\sin(T2)\sin(T3)\big) - \sin(T4)\big(\cos(T1)\cos(T2)\sin(T3) + \cos(T1)\sin(T2)\big)\big) - \sin(T5)\sin(T1)$$

$$R_{12} = -\sin(T5)\big(\cos(T4)\big(\cos(T1)\cos(T2)\cos(T3) - \cos(T1)\sin(T2)\sin(T3)\big) - \sin(T4)\big(\cos(T1)\cos(T2)\sin(T3) + \cos(T1)\sin(T2)\big)\big) - \cos(T5)\sin(T1)$$

$$R_{13} = -\cos(T1)\sin(T2 + T3 + T4)$$

$$R_{21} = \cos(T5)\big(\cos(T4)\big(\cos(T2)\cos(T3)\sin(T1) - \sin(T1)\sin(T2)\sin(T3)\big) - \sin(T4)\big(\cos(T2)\sin(T1)\sin(T3) + \cos(T3)\sin(T1)\sin(T2)\big)\big)$$

$$R_{22} = \cos(T1)\cos(T5) - \sin(T5)\big(\cos(T4)\big(\cos(T2)\cos(T3)\sin(T1) - \sin(T1)\sin(T2)\sin(T3)\big) - \sin(T4)\big(\cos(T2)\sin(T1)\sin(T3) + \cos(T3)\sin(T1)\sin(T2)\big)\big)$$

$$R_{23} = \cos(T1 + T2 + T3 + T4)/2 - \cos(T2 - T1 + T3 + T4)/2$$

$$R_{31} = \cos(T5)\sin(T2 + T3 + T4)$$

$$R_{32} = -\sin(T5)\sin(T2 + T3 + T4)$$

Se obtienen las siguientes posiciones:

$$dx = \cos(T1)\big(10\cos(T2 + T3) + 15\cos(T2) - \frac{181\sin(T2 + T3 + T4)}{10}\big)$$

$$dy = \sin(T1)\big(10\cos(T2 + T3) + 15\cos(T2) - \frac{181\sin(T2 + T3 + T4)}{10}\big)$$

$$dz = 10\sin(T2 + T3) + \frac{181\cos(T2 + T3 + T4)}{10} + 15\sin(T2) + 12$$

5.2 Cinemática Inversa

La cinemática inversa (IK) consiste en encontrar los ángulos de las articulaciones $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$ 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 (O_x, O_y, O_z) y la orientación final (que nos provee el vector director del eje z del efector, dado por (r_{13}, r_{23}, r_{33}), se calcula la posición del centro de la muñeca restando la componente a lo largo del último eje:

$$x_c = O_x - d_6 r_{13}, \quad y_c = O_y - d_6 r_{23}, \quad z_c = O_z - d_6 r_{33}.$$

Se asume que el último eslabón (con longitud d_6) 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.

3. **Determinación de $\boldsymbol{\theta_1, \theta_2, \theta_3}$** :
   Conocido el centro de la muñeca (x_c, y_c, z_c), 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 $\boldsymbol{\theta_1}$**:
     $\theta 1 = arctan2(y_c, x_c)$.

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 $\boldsymbol{\theta_3}$** (el codo) y **$\boldsymbol{\theta_2}$** (el hombro):

$$D = \frac{x_c^2 + y_c^2 + (z_c - 12)^2 - 15^2 - 10^2}{2 \cdot 15 \cdot 10}$$

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 (θ_3) es una práctica estándar en cinemática inversa para robots con geometría tipo RR (rotacional-rotacional en el plano). Dicho de forma genérica:

$$\cos(\theta_3) = \frac{L_2^2 + L_3^2 - R^2}{2L_2L_3}$$

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 θ_3.

Una vez calculado θ_3, para θ_2 se utilizan relaciones trigonométricas adicionales:

$$\theta_2 = \arctan 2((Z_c - 12), \sqrt{x_c^2 + y_c^2}) - \arctan 2(10\sin(\theta_3), 15 + 10\cos(\theta_3))$$

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 $\theta_1, \theta_2, \theta_3$.

4. **Cálculo de θ_4 y θ_5**:
   Una vez se tienen los primeros 3 ángulos, se puede calcular la orientación de la muñeca aislando la rotación final. Esto se hace con:

$$R = R_3^0 R_5^3 \implies R_5^3 = (R_3^0)^{-1} R$$

donde R es la orientación deseada final del efector, R_3^0 es la orientación de la base hasta la articulación 3 (calculada a partir de $\theta_1, \theta_2, \theta_3$) y R_5^3 es la rotación pura debida a las articulaciones 4 y 5.

$$N = R_{33} \cdot \cos(\theta_2 + \theta_3) - R_{13} \cdot \sin(\theta_2 + \theta_3) \cdot \cos(\theta_1) - R_{23} \cdot \sin(\theta_2 + \theta_3) \cdot \sin(\theta_1)$$

$$\theta_4 = \text{atan2}(\pm\sqrt{1 - N^2}, N)$$

$$M = -(R_{12} \cdot \sin(\theta_1) - R_{22} \cdot \cos(\theta_1))$$

$$\theta_5 = \text{atan2}(\pm\sqrt{1 - M^2}, M)$$

Estas fórmulas para θ_4 y θ_5 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 θ_4, θ_5 a partir de la submatriz R_5^3.

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, θ_4 y θ_5.

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

```
// arduino_test.ino

#include <Servo.h>

// Definir los objetos Servo para cada uno de los 6 servos
Servo servo1, servo2, servo3, servo4, servo5, servo6;

// Asignar los pines a cada servo
const int pinServo1 = 3;
const int pinServo2 = 5;
const int pinServo3 = 6;
const int pinServo4 = 9;
const int pinServo5 = 10;
const int pinServo6 = 11;

void setup() {
  // Iniciar la comunicación serial a 9600 bps
  Serial.begin(9600);

  // Adjuntar cada servo a su pin correspondiente
  servo1.attach(pinServo1);
  servo2.attach(pinServo2);
  servo3.attach(pinServo3);
  servo4.attach(pinServo4);
  servo5.attach(pinServo5);
  servo6.attach(pinServo6);

  // Posición inicial para todos los servos
  servo1.write(180);
  servo2.write(165);
  servo3.write(90);
  servo4.write(90);
  servo5.write(90);
  servo6.write(130);

  // Confirmar inicio
  Serial.println("Arduino listo para recibir comandos.");
}

void loop() {
  // Verificar si hay datos disponibles en el buffer serial
  if (Serial.available() > 0) {
    // Leer la línea completa hasta el carácter de nueva línea
    String command = Serial.readStringUntil('\n');
    command.trim(); // Eliminar espacios en blanco adicionales

    // Comandos esperados:
    // "MOVE t1 t2 t3 t4 t5 t6"
    // "GRIP OPEN"
    // "GRIP CLOSE"

    if (command.startsWith("MOVE")) {
      // Dividir el comando por espacios
      int t1, t2, t3, t4, t5, t6;
      sscanf(command.c_str(), "MOVE %d %d %d %d %d %d", &t1, &t2, &t3, &t4, &t5, &t6);

      // 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

```python
# test_control.py

import serial
import time
import tkinter as tk
from tkinter import ttk, messagebox
import threading


class ServoController:
    def __init__(self, port='COM6', baudrate=9600, timeout=1):
        try:
            self.ser = serial.Serial(port, baudrate, timeout=timeout)
            time.sleep(2)  # Esperar a que el Arduino reinicie
            print(f"Conectado a {port} a {baudrate} bps.")
            self.read_serial()  # Leer el mensaje de inicio
        except serial.SerialException as e:
            print(f"Error al conectar con el puerto serial: {e}")
            self.ser = None

    def send_command(self, command):
        if self.ser and self.ser.is_open:
            try:
                self.ser.write((command + '\n').encode())
                print(f"Enviado: {command}")
                # Leer respuesta no bloqueante
                response = self.read_serial()
                return response
            except serial.SerialException as e:
                print(f"Error al enviar comando: {e}")
                return None
        else:
            print("Puerto serial no está abierto.")
            return None

    def read_serial(self):
        if self.ser and self.ser.in_waiting > 0:
            try:
                response = self.ser.readline().decode().strip()
                if response:
                    print(f"Arduino: {response}")
                    return response
            except serial.SerialException as e:
                print(f"Error al leer del serial: {e}")
        return None

    def move_servos(self, angles):
        """
        angles: lista o tupla de 6 ángulos (t1, t2, t3, t4, t5, t6)
        """
```

```
        if len(angles) != 6:
            print("Debe proporcionar exactamente 6 ángulos.")
            return None
        command = f"MOVE {' '.join(map(str, angles))}"
        return self.send_command(command)

    def grip_open(self):
        return self.send_command("GRIP OPEN")

    def grip_close(self):
        return self.send_command("GRIP CLOSE")

    def close(self):
        if self.ser and self.ser.is_open:
            self.ser.close()
            print("Puerto serial cerrado.")


class ServoGUI:
    def __init__(self, root, controller: ServoController):
        self.root = root
        self.controller = controller
        self.root.title("Control de Servos con Arduino")
        self.create_widgets()
        # self.current_angles = [90] * 6  # Ángulos iniciales
        self.current_angles = [180,165, 90, 90, 90, 180]
        self.update_log_thread = threading.Thread(target=self.update_log, daemon=True)
        self.update_log_thread.start()

    def create_widgets(self):
        # Crear un frame para los sliders
        sliders_frame = ttk.LabelFrame(self.root, text="Control de Servos")
        sliders_frame.pack(padx=10, pady=10, fill="x")

        self.sliders = []
        self.slider_vars = []
        self.angle_labels = []

        for i in range(6):
            var = tk.IntVar(value=90)
            self.slider_vars.append(var)
            slider = ttk.Scale(
                sliders_frame,
                from_=0,
                to=180,
                orient='horizontal',
                variable=var,
                command=lambda val, idx=i: self.update_label(idx)
            )
            slider.grid(row=i, column=1, padx=5, pady=5, sticky="ew")
            sliders_frame.columnconfigure(1, weight=1)

            label = ttk.Label(sliders_frame, text=f"Servo {i + 1}:")
            label.grid(row=i, column=0, padx=5, pady=5, sticky="w")

            angle_label = ttk.Label(sliders_frame, text="90")
            angle_label.grid(row=i, column=2, padx=5, pady=5, sticky="w")
            self.sliders.append(slider)
            self.angle_labels.append(angle_label)

        # Botón para mover el brazo
        move_button = ttk.Button(self.root, text="Mover Servos",
command=self.move_servos_smooth)
        move_button.pack(pady=5)

```

```
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
```

```
        contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        rois = [cv2.boundingRect(cnt) for cnt in contours if (
            MIN_WIDTH < cv2.boundingRect(cnt)[2] < MAX_WIDTH and
            MIN_HEIGHT < cv2.boundingRect(cnt)[3] < MAX_HEIGHT and
            cv2.contourArea(cnt) > MIN_AREA and
            cv2.contourArea(cnt) < MAX_AREA)]
        rois = sorted(rois, key=lambda item: item[0])

        # Lista para almacenar números detectados
        numbers = []

        for bbox in rois:
            prepared_img = preprocess_image(thresh, bbox)
            if prepared_img is not None:
                number = self.classifier.predict_number(prepared_img)
                if number is not None and 0 <= number <= 9:
                    numbers.append(number)
                    # Dibujar rectángulo y número detectado
                    x, y, w, h = bbox
                    cv2.rectangle(roi_global, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    cv2.putText(roi_global, str(number), (x, y - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
        return thresh

def preprocess_image(thresh, bbox, img_width=28, img_height=28):
    """Extrae, invierte, redimensiona y normaliza la ROI individual para la
predicción."""
    x, y, w, h = bbox
    margin = 5
    y_start = max(y - margin, 0)
    y_end = min(y + h + margin, thresh.shape[0])
    x_start = max(x - margin, 0)
    x_end = min(x + w + margin, thresh.shape[1])
    roi = thresh[y_start:y_end, x_start:x_end]
    if roi.size == 0:
        return None
    roi = cv2.bitwise_not(roi)
    try:
        resized = cv2.resize(roi, (img_width, img_height))
    except Exception as e:
        print(f"Error resizing ROI: {e}")
        return None
    normalized = resized.astype('float32') / 255.0
    reshaped = np.expand_dims(normalized, axis=-1)
    reshaped = np.expand_dims(reshaped, axis=0)
    return reshaped

class RobotController:
    def __init__(self, servo_controller: ServoController, classifier: NumberClassifier,
video_stream: VideoStream, predicted_number_var):
        self.servo = servo_controller
        self.classifier = classifier
        self.video_stream = video_stream
        self.cap = video_stream.cap  # Usar la misma captura de video
        self.predicted_number_var = predicted_number_var  # Variable para actualizar el
número predicho en la GUI
        # Definir las posiciones predefinidas
        self.positions = {
            "initial": [180, 165, 90, 90, 90, 150],
            "object_safe": [150, 165, 100, 20, 90, 130],
            "object_pick": [150, 90, 130, 35, 90, 180],
            "object_pick_GC": [150, 90, 130, 35, 90, 130],
            "camera_safe": [180, 165, 100, 50, 90, 130],
            "camera_show": [180, 140, 100, 50, 90, 130],
```

```
            # Posiciones de clasificación Safe
            "classification_A_safe": [100, 165, 100, 50, 90, 130],
            "classification_B_safe": [70, 165, 100, 50, 90, 130],
            "classification_C_safe": [40, 165, 100, 50, 90, 130],
            "classification_D_safe": [10, 165, 100, 50, 90, 130],
            # Posiciones de clasificación Put
            "classification_A_put": [100, 120, 150, 50, 90, 130],
            "classification_B_put": [70, 120, 150, 50, 90, 130],
            "classification_C_put": [40, 120, 150, 50, 90, 130],
            "classification_D_put": [10, 120, 150, 50, 90, 130]
        }

        # Definir el mapeo de números a posiciones de clasificación
        self.number_to_classification = {
            0: "A",
            1: "A",
            2: "A",
            3: "B",
            4: "B",
            5: "B",
            6: "C",
            7: "C",
            8: "D",
            9: "D"
        }

        # Parámetros de ROI para la clasificación (ajusta según sea necesario)
        self.roi_params = (260, 150, 150, 150)  # (GLOBAL_X, GLOBAL_Y, GLOBAL_W, GLOBAL_H)
        self.detection_params = (5, 150, 50, 150, 200, 4000)  # (MIN_WIDTH, MAX_WIDTH, MIN_HEIGHT, MAX_HEIGHT, MIN_AREA, MAX_AREA)

        # Inicializar ángulos actuales
        self.current_angles = self.positions["initial"].copy()
        self.servo.move_servos(self.current_angles)  # Asegurar que el robot comience en la posición inicial

    def smooth_move(self, target_angles, step=1, delay=0.02):
        """
        Realiza un movimiento suave desde los ángulos actuales hasta los ángulos objetivo.

        Args:
            target_angles (list): Lista de 6 ángulos objetivo.
            step (int): Incremento/decremento por paso.
            delay (float): Tiempo en segundos entre pasos.
        """
        max_steps = max(abs(t - c) for c, t in zip(self.current_angles, target_angles))
        steps = max_steps // step if step > 0 else 1

        if steps == 0:
            steps = 1

        new_angles = self.current_angles.copy()

        for _ in range(steps):
            for i in range(6):
                if new_angles[i] < target_angles[i]:
                    new_angles[i] = min(new_angles[i] + step, target_angles[i])
                elif new_angles[i] > target_angles[i]:
                    new_angles[i] = max(new_angles[i] - step, target_angles[i])
            self.servo.move_servos(new_angles)
            self.current_angles = new_angles.copy()
            time.sleep(delay)

```

```
        # Asegurarse de que los ángulos finales sean exactamente los objetivos
        self.servo.move_servos(target_angles)
        self.current_angles = target_angles.copy()

    def execute_sequence(self):
        threading.Thread(target=self._sequence_thread, daemon=True).start()

    def _sequence_thread(self):
        try:
            # 1. Mover a Posición Inicial
            print("Moviendo a Posición Inicial")
            self.smooth_move(self.positions["initial"])
            time.sleep(0.5)  # Esperar a que el movimiento se complete

            # 2. Mover a Posición Safe del Objeto
            print("Moviendo a Posición Safe del Objeto")
            self.smooth_move(self.positions["object_safe"])
            time.sleep(0.5)

            # 3. Mover a Posición Pick para recoger el objeto
            print("Moviendo a Posición Pick")
            self.smooth_move(self.positions["object_pick"])
            time.sleep(0.5)

            print("Moviendo a Posición Pick GC")
            self.smooth_move(self.positions["object_pick_GC"])
            time.sleep(0.5)

            # 4. Cerrar el Gripper
            print("Cerrando Gripper")
            self.servo.grip_close()
            time.sleep(1)  # Esperar a que el gripper cierre

            # 5. Mover a Posición Safe del Objeto
            print("Moviendo de vuelta a Posición Safe del Objeto")
            self.smooth_move(self.positions["object_safe"])
            time.sleep(0.5)

            # 6. Mover a Posición de la Cámara Safe
            print("Moviendo a Posición de la Cámara Safe")
            self.smooth_move(self.positions["camera_safe"])
            time.sleep(0.5)

            # 7. Mover a Posición de la Cámara Show
            print("Moviendo a Posición de la Cámara Show")
            self.smooth_move(self.positions["camera_show"])
            time.sleep(1)  # Esperar a que el servo se estabilice

            # 8. Esperar para ajustar la cámara antes de capturar
            print("Ajusta la cámara. La detección se realizará en 5 segundos.")
            time.sleep(5)

            # 9. Capturar y Clasificar el Número
            print("Capturando imagen para clasificación")
            ret, frame = self.cap.read()
            if not ret:
                print("Error: No se pudo capturar la imagen.")
                messagebox.showerror("Error", "No se pudo capturar la imagen.")
                return

            number = get_number_from_frame(frame, self.classifier, self.roi_params,
self.detection_params)
            if not number:
                print("No se detectó un número válido.")
                messagebox.showerror("Error", "No se detectó un número válido.")
```

```
                self.predicted_number_var.set("No se detectó un número válido.")
                return

            # Asumimos que solo se detecta un número por pieza
            detected_number = number[0]
            print(f"Número detectado: {detected_number}")
            self.predicted_number_var.set(f"Número detectado: {detected_number}")

            # Obtener la clasificación correspondiente
            classification_label = self.number_to_classification.get(detected_number,
None)
            if classification_label is None:
                print(f"Número {detected_number} no tiene asignada una ubicación de
clasificación.")
                messagebox.showerror("Error", f"Número {detected_number} no tiene
asignada una ubicación de clasificación.")
                return

            # Determinar las claves de posición para la clasificación
            classification_safe_key = f"classification_{classification_label}_safe"
            classification_put_key = f"classification_{classification_label}_put"

            if classification_safe_key in self.positions and classification_put_key in
self.positions:
                print(f"Moviendo a Posición Safe para Clasificación
{classification_label}")
                self.smooth_move(self.positions[classification_safe_key])
                time.sleep(0.5)

                print(f"Moviendo a Posición Put para Clasificación
{classification_label}")
                self.smooth_move(self.positions[classification_put_key])
                time.sleep(0.5)

                # Abrir el Gripper para soltar la pieza
                print("Abriendo Gripper")
                self.servo.grip_open()
                time.sleep(1)  # Esperar a que el gripper abra

                # Volver a la Posición Safe de Clasificación
                print(f"Volviendo a Posición Safe para Clasificación
{classification_label}")
                self.smooth_move(self.positions[classification_safe_key])
                time.sleep(0.5)
            else:
                print(f"No hay posición definida para la clasificación
{classification_label}.")
                messagebox.showerror("Error", f"No hay posición definida para la
clasificación {classification_label}.")
                return

            # 10. Volver a Posición Inicial
            print("Volviendo a Posición Inicial")
            self.smooth_move(self.positions["initial"])
            time.sleep(0.5)

            messagebox.showinfo("Éxito", f"Pieza clasificada y colocada en la posición
{classification_label}.")

        except Exception as e:
            print(f"Error durante la secuencia: {e}")
            messagebox.showerror("Error", f"Error durante la secuencia: {e}")

class CustomServoGUI(tk.Frame):
    """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.

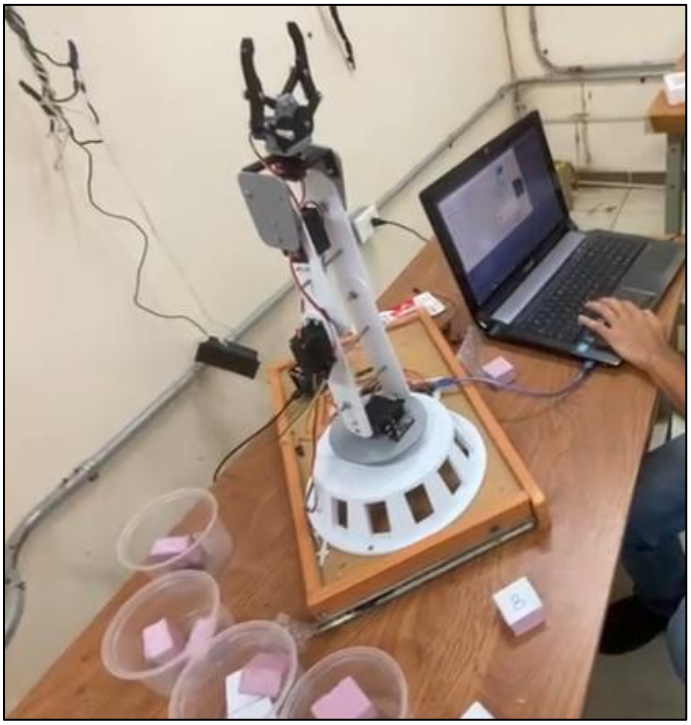

Figura 8. Brazo robótico en posición inicial.

2. **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.

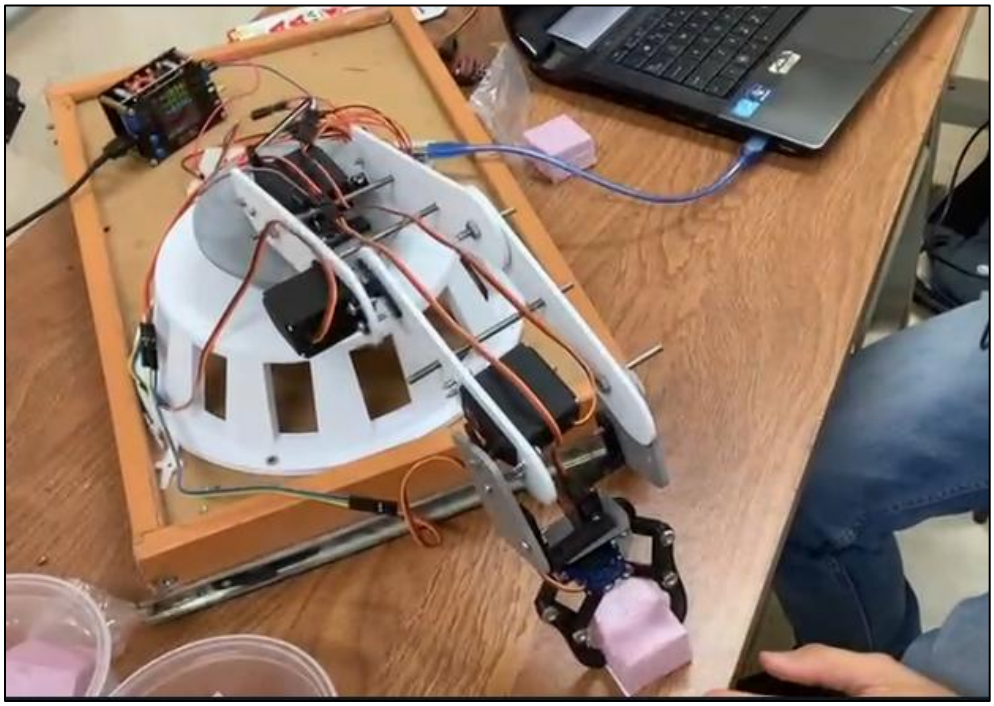

Figura 9. Brazo robótico agarrando el objeto.

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

Figura 10. Brazo robótico mostrando el objeto a la webcam,

4. **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.

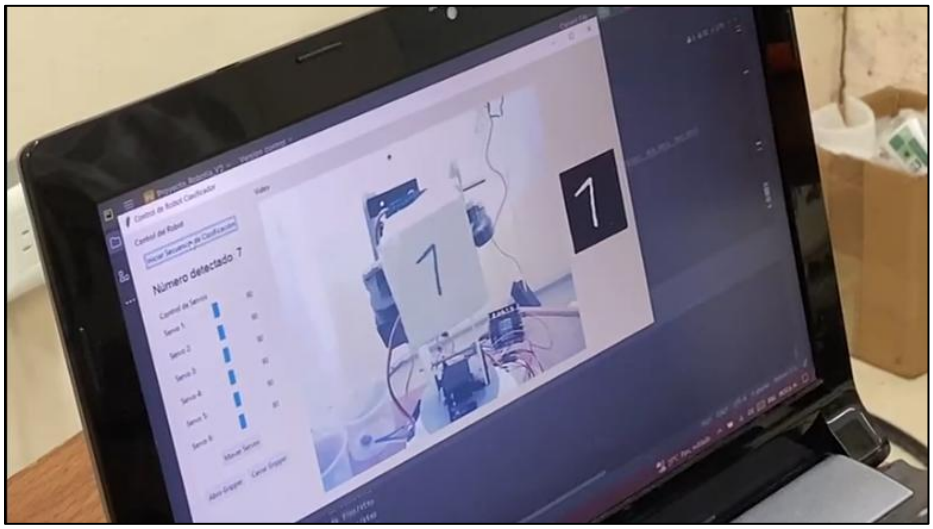

Figura 11. Procesamiento y clasificación de la imagen capturada.

5. **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 12. 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