Brazo Robótico de 4GDL con Visión Artificial

Juan Pablo Gasca Calderón, Hebert Elías Palmera Buelvas, Diego Fernando Gonzalez Ruiz
Corporación Universitaria del Huila - CORHUILA
Neiva-Huila, Colombia
jpgasca2019-2@corhuila.edu.co
dfgonzalez2019-2@corhuila.edu.co
hepalmera2019-2@corhuila.edu.co

***Abstract*— Este informe aborda el diseño y funcionamiento de un brazo robótico de 4 grados de libertad (4GDL) equipado con visión artificial para la clasificación de medicamentos en la industria farmacéutica. Se destacan los componentes electrónicos utilizados, como servomotores y placas de desarrollo Arduino, y se describe el proceso de diseño mecánico en Solidworks. Se presenta la implementación de la cinemática inversa del robot mediante MATLAB y se detallan los códigos para diseñar trayectorias de movimiento en función de figuras geométricas. Se enfatiza la importancia de la visión computarizada a través de OpenCV para la clasificación de envases farmacéuticos. En última instancia, se subraya el potencial de esta tecnología para mejorar la eficiencia y precisión en la cadena de suministro farmacéutica.**

I. INTRODUCCIÓN

La evolución de la tecnología ha permitido que la combinación de la robótica y la visión artificial transforme la forma en que abordamos tareas críticas en diversas industrias. En el ámbito farmacéutico, la precisión y la eficiencia en la clasificación de medicamentos son esenciales para garantizar la salud y el bienestar de la sociedad. En este contexto, la implementación de un brazo robótico de 4 grados de libertad (4GDL) equipado con un sistema de visión artificial representa un avance significativo en la tecnología aplicada.

El objetivo principal de este estudio es proporcionar una visión detallada del diseño y funcionamiento de un brazo robótico de 4GDL elaborado específicamente para la clasificación de medicamentos, utilizando visión artificial para el reconocimiento de patrones geométricos, tales como cuadrados, rectángulos y círculos. Este sistema no solo es un ejemplo de cómo la automatización y la inteligencia artificial se fusionan para abordar desafíos complejos, sino que también demuestra su capacidad de aplicación en la industria farmacéutica. Además, se exploran en profundidad las características técnicas del brazo robótico, la creación de trayectorias de movimiento, el procesamiento de imágenes mediante algoritmos y la simulación de los movimientos necesarios para la clasificación de medicamentos.

A medida que avanzamos hacia un futuro en el que la automatización y la visión artificial se convierten en pilares fundamentales de la innovación, la comprensión de la influencia de esta tecnología en la industria farmacéutica y en la mejora de los procesos de clasificación se vuelve imperativa. Este informe analiza el potencial de un brazo robótico de 4GDL equipado con visión artificial para transformar la clasificación de medicamentos. Esta combinación de tecnologías tiene el potencial de mejorar la eficiencia, la precisión y la seguridad de la cadena de suministro farmacéutica..

II. MATERIALES Y MÉTODOS

A. Planteamiento del Problema

En la industria farmacéutica, la precisión y la eficiencia en la clasificación y manipulación de productos son de vital importancia para garantizar la calidad y la seguridad de los medicamentos. La clasificación de productos implica la identificación y separación de productos farmacéuticos basados en sus características específicas, como tamaño, forma, etiquetado y contenido. Actualmente, esta tarea se realiza en gran medida a través de la mano de obra humana y sistemas de automatización convencionales, lo que conlleva limitaciones en términos de velocidad, precisión y flexibilidad.

Por ende, la implementación de un brazo robótico de 4GDL para la tarea de "pick and place" en la clasificación de productos en la industria farmacéutica surge como una solución potencial para abordar estos desafíos. Sin embargo, la adopción exitosa de esta tecnología implica considerar una serie de factores críticos, como el diseño del sistema, la programación de la inteligencia artificial, la integración con sistemas de visión y seguimiento, y la validación de procesos conforme a las regulaciones de la industria..

B. Diseño Electrónico

La creación del brazo robótico de 4GDL requirió la implementación de los siguientes componentes electrónicos esenciales, los cuales fueron fundamentales para su construcción a escala.

1. Servomotor MG995

Actuador de rotación continua con un destacado par de torsión de aproximadamente 9,4 kg·cm (4,8–6,6 V) y un rango de rotación de 180 grados. Controlado mediante señales PWM, este servomotor funciona dentro de un rango de voltaje de 4,8 V a 6,6 V, ofreciendo una velocidad de rotación que varía según el voltaje de alimentación. Su construcción robusta con una carcasa metálica y rodamientos de bolas lo hace adecuado para aplicaciones de robótica, modelismo y automatización que requieren precisión en el control de posición y una alta fuerza de torsión.

Figura 1. Servomotor MG995

2. Arduino Mega 2560

Placa de desarrollo basada en el microcontrolador ATmega2560 con 256 KB de memoria Flash, 8 KB de SRAM y 4 KB de EEPROM. Ofrece 54 pines digitales, 16 entradas analógicas y

múltiples opciones de comunicación. Su versatilidad lo hace idóneo para proyectos avanzados que requieren una amplia E/S, mayor capacidad de procesamiento y aplicaciones complejas, como robótica, automatización y sistemas de control.

Figura 2. Arduino Mega 2560

3. Adaptador de voltaje

Dispositivo de suministro de energía diseñado para proporcionar una tensión constante de 5 voltios y una corriente máxima de 3 amperios. Esto implica que puede entregar una potencia máxima de 15 vatios (5V x 3A). Estos adaptadores son comunes y se utilizan para alimentar una variedad de dispositivos electrónicos, como cargadores de teléfonos móviles, placas de desarrollo, dispositivos USB y otros equipos que requieren una fuente de alimentación estable a 5 voltios con la capacidad de suministrar hasta 3 amperios de corriente.

Figura 3. Adaptador 5V a 3A.

Implementando los componentes mencionados, a continuación se presenta el esquema de conexiones para asegurar el funcionamiento adecuado del robot. Es esencial considerar los pines de señal PWM para los servomotores, en este caso, se emplearon los pines digitales desde D9 hasta D13

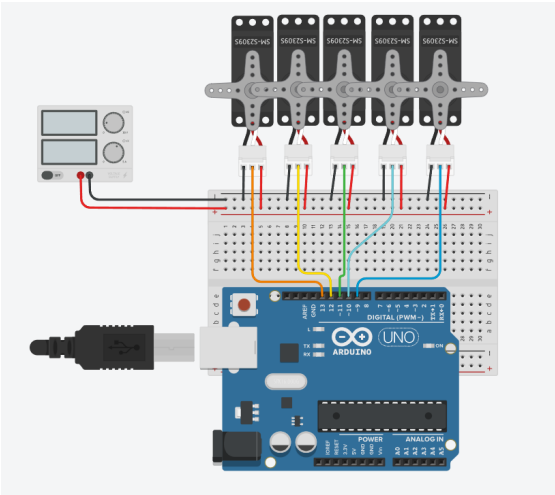


Figura 4. Circuito esquemático de conexiones.

C. Diseño mecánico en Solidworks

La etapa de diseño mecánico en Solidworks desempeña un papel esencial en el desarrollo de un brazo robótico, ya que permite evaluar y perfeccionar el concepto antes de invertir recursos significativos en su construcción física. A través de Solidworks, se pueden crear modelos 3D precisos y detallados que representan fielmente el diseño propuesto, lo que brinda la capacidad de simular su funcionamiento, analizar la resistencia de los componentes, verificar la cinemática y dinámica del brazo, y realizar pruebas de viabilidad. Esto no solo ahorra tiempo y recursos al identificar y corregir posibles problemas antes de la fabricación, sino que también permite una optimización más efectiva del diseño, lo que se traduce en una mayor eficiencia y rendimiento del brazo robótico en su operación real. En última instancia, el diseño en Solidworks garantiza que la implementación física sea más precisa, segura y económica, al tiempo que reduce los riesgos asociados a errores de diseño y fabricación.

Por esta razón, se optó por la utilización del software de diseño asistido por computadora SOLIDWORKS para llevar a cabo el diseño y ensamblaje de las piezas esenciales en la creación del brazo robótico de 4 grados de libertad (4GDL). El conjunto de piezas diseñadas engloba elementos como el servo bracket long U (3), que conforma la base del brazo robótico, el servo bracket shaped (4), desempeñando el papel de soporte mecánico para los servomotores, los servo disc (5), que actúan como ejes de los servomotores. Además, se incorporan los servo bracket U (4) para ensamblar los eslabones del brazo, el servo bracket corner mount y diversos accesorios, incluyendo tornillos, tuercas y bujes.

Figura 5. Servo Bracket Long U

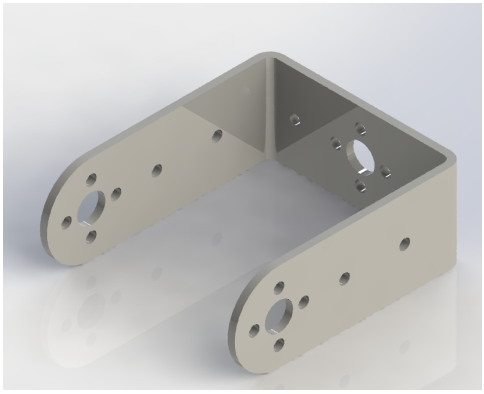


Figura 6. Servo Bracket U

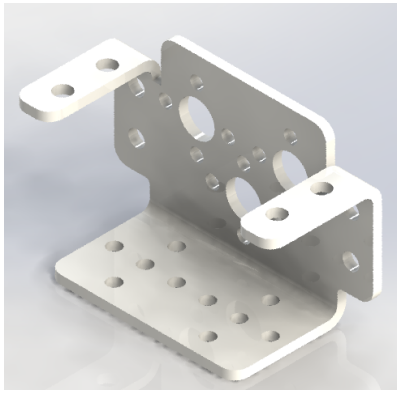


Figura 7. Servo Bracket Shaped

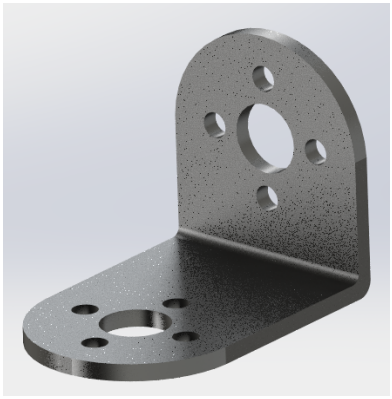


Figura 8. Servo Bracket Corner Mount

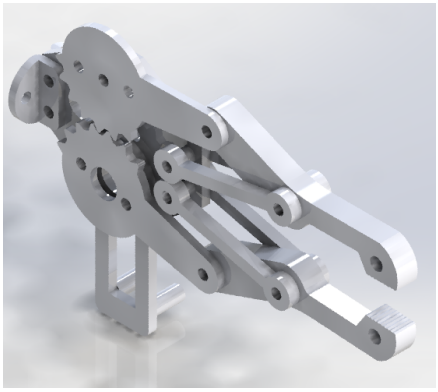


Figura 9. Gripper

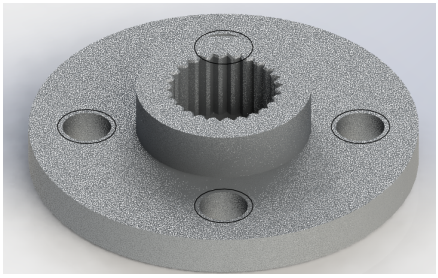


Figura 10. Servo Disk

Una vez que se completaron las piezas, se procedió al ensamblaje del robot con el objetivo de obtener una representación previa de cómo se vería el sistema robótico a escala real. Dado que SolidWorks cuenta con un módulo de complementos específico, en particular, Photoview 3D, esta herramienta se utilizó para lograr una representación realista del robot en el proceso de renderizado.

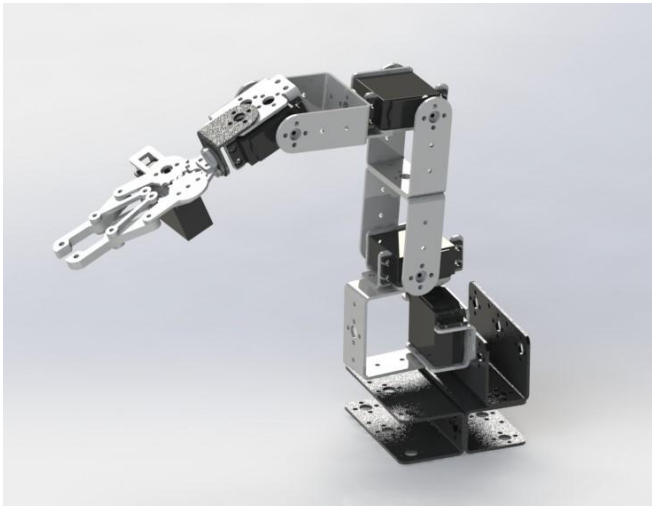


Figura 11. Ensamblaje del Brazo Róbotico 4GDL - Renderizado

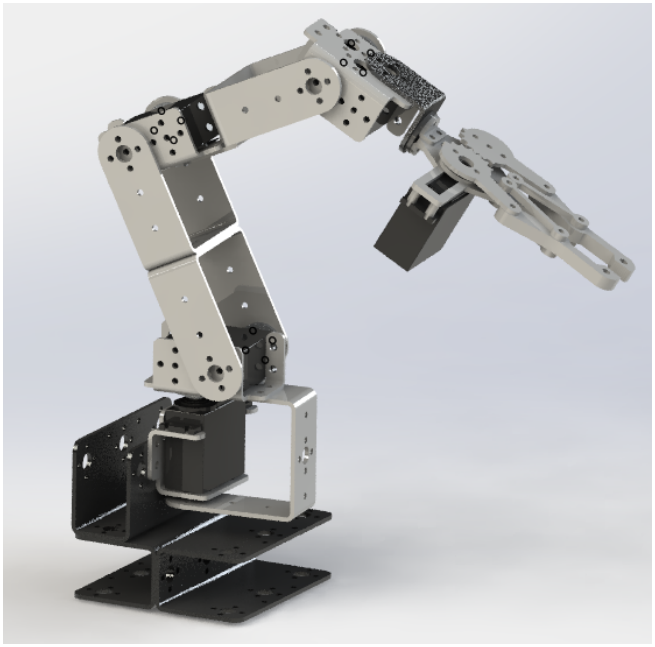


Figura 12. Ensamblaje del Brazo Róbotico 4GDL - Renderizado

D. Definición de Trayectorias en Matlab

Las trayectorias espaciales del robot se diseñaron utilizando un enfoque de cinemática inversa de movimiento, que consiste en un conjunto de ecuaciones matemáticas que determinan los ángulos de rotación necesarios en las articulaciones para alcanzar una ubicación específica del efector final en el espacio. En otras palabras, estas ecuaciones permiten traducir con precisión la posición exacta del extremo del robot en el espacio en los valores de ángulo requeridos en cada articulación.

La cinemática inversa del robot se desarrolló mediante el uso del software MATLAB, que posibilitó la simulación del brazo robótico y la determinación de los puntos precisos que el robot debía alcanzar, teniendo en consideración su espacio de trabajo.

A continuación se presentan los códigos implementados para la creación de las trayectorias del robot. El primer fragmento de

código corresponde a una función llamada "CinInv4GDL_3D" que calcula la cinemática inversa para un brazo robótico de 4 grados de libertad (4GDL) en un entorno tridimensional. Esta función toma las coordenadas (x, y, z) de la posición deseada del efector final del robot, así como los parámetros adicionales "qp" y "codo". La función calcula los ángulos de las articulaciones q1, q2, q3 y q4 necesarios para colocar el efector final en la posición y orientación especificada.

Código de la Cinemática Inversa (4GDL):

```
%% Cinematica Inversa de 4GDL - 3D
function[q1,q2,q3,q4] = CinInv4GDL_3D(x,y,z,qp,codo)

L1 = 0.095;
u = sqrt(x^2+y^2);
q1 = atan(y/x);

[q2,q3,q4] = CinInv3GDL(u,z-L1,qp, codo);

end
```

Cabe destacar que este código utilizó códigos previos de cinemáticas inversas vistas en clase, tales cómo la cinemática inversa 3GDL y 2GDL. El siguiente código se encarga de calcular la cinemática inversa para un brazo robótico con 3 grados de libertad (3GDL). Recibe como entrada las coordenadas (x, z) de la posición deseada en el plano horizontal, así como el ángulo qy y el parámetro "codo" que determina la orientación del codo del robot. El código primero ajusta las coordenadas en función del ángulo qy y luego llama a una función de cinemática inversa de 2 grados de libertad (CinInv2GDL) para calcular los ángulos q1 y q2 necesarios en la base del brazo. Luego, se calcula q3, que es el ángulo del tercer grado de libertad del brazo robótico.

Código de la Cinemática Inversa (3GDL):

```
function [q1,q2,q3]=CinInv3GDL(x,z,qy,codo)

L3=0.155; %L4

xp=x-L3*cos(qy);
zp=z-L3*sin(qy);

[q1,q2]=CinInv2GDL(xp,zp,codo);

q3=qy-q1-q2;
end
```

Para el caso de tener un robot de 2GDL se implementa el siguiente código donde se deben proporcionar las coordenadas (x, z) en el plano horizontal y el parámetro "codo", que determina si el codo del robot está arriba o abajo. El código utiliza fórmulas trigonométricas para calcular los ángulos q1 y q2 necesarios para posicionar el brazo en la posición deseada. Dependiendo de si el codo del robot está arriba o abajo, se calculan los ángulos de manera diferente.

Código de la Cinemática Inversa (2GDL):

```
function [q1,q2]=CinInv2GDL(x,z,codo)

disp('Codo abajo = 1, Codo arriba otro')
L1=0.105; %l2
L2=0.098; %l3

alfa=atan(z/x);
 r=sqrt(x^2+z^2);
beta=acos((L1^2+r^2-L2^2)/(2*L1*r));
gamma=acos((L1^2-r^2+L2^2)/(2*L1*L2));

if codo==1

%Codo abajo
q1=alfa-beta;
q2=pi-gamma;
else
%Codo arriba
q1=alfa+beta;
q2=-(pi-gamma);
end
```

El segundo fragmento de código se encarga de definir trayectorias de movimiento para el robot con base en el tipo de figura que se quiere clasificar (cuadrado, rectángulo o círculo). El usuario ingresa el tipo de figura, y en función de esa entrada, se establecen las coordenadas iniciales y finales para el robot. Luego, se calculan una serie de puntos intermedios a lo largo de una trayectoria lineal entre las coordenadas iniciales y finales. Finalmente, se utiliza la función de cinemática inversa "***CinInv4GDL_3D***" para determinar los ángulos de las articulaciones del robot en cada punto intermedio, lo que permite al robot seguir la trayectoria lineal deseada en el espacio tridimensional. La visualización gráfica de la trayectoria se muestra en un gráfico 3D.

Código Trayectoria de los 3 tipos de clasificación:

```
% Solicitar al usuario que ingrese el tipo de figura
tipo_figura = input('Ingrese el tipo de figura (cuadrado, rectangulo
o circulo): ', 's');

% Definir los puntos iniciales y finales en función del tipo de
figura
if strcmp(tipo_figura, 'cuadrado')
    x_inicial = 0.2;
    y_inicial = -0.1;
    z_inicial = 0;

    x_final = 0.18;
    y_final = 0.1;
    z_final = 0.2;
elseif strcmp(tipo_figura, 'rectangulo')
    x_inicial = 0.2;
    y_inicial = -0.1;
    z_inicial = 0;

    x_final = 0.2;
    y_final = 0.1;
    z_final = 0.2;
elseif strcmp(tipo_figura, 'circulo')
    x_inicial = 0.2;
    y_inicial = -0.1;
    z_inicial = 0;

    x_final = 0.22;
    y_final = 0.1;
    z_final = 0.2;
else
     error('Tipo de figura no válido. Debe ser cuadrado, rectangulo o
circulo.');
end
```

```
% Número de puntos intermedios
n = 40;

% Calcular coordenadas intermedias
x_intermedias = linspace(x_inicial, x_final, n);
y_intermedias = linspace(y_inicial, y_final, n);
z_intermedias = linspace(z_inicial, z_final, n);

% Cinemática inversa y cálculo de articulaciones
q1 = zeros(1, n);
q2 = zeros(1, n);
q3 = zeros(1, n);
q4 = zeros(1, n);

for i = 1:n
    [q1(i), q2(i), q3(i), q4(i)] = CinInv4GDL_3D(x_intermedias(i),
y_intermedias(i), z_intermedias(i), 0, 0);
end

% Visualización de la trayectoria lineal
figure;
hold on;
plot3(x_intermedias, y_intermedias, z_intermedias, 'or');
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Trayectoria Lineal del Robot');
```

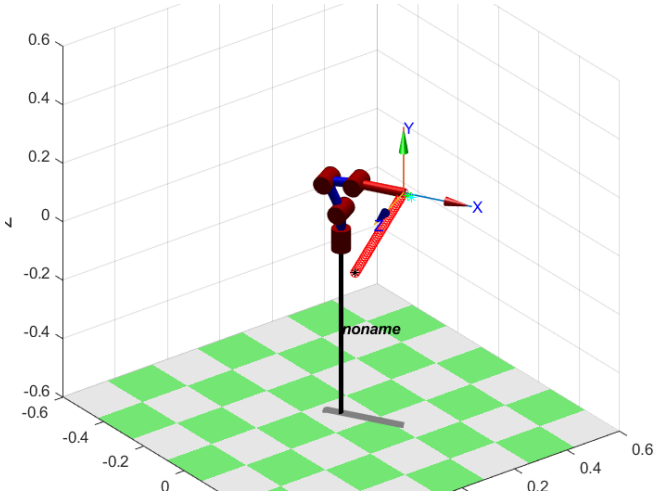
Después de completar los pasos anteriores, se llevó a cabo una simulación tridimensional del robot con el fin de identificar las ubicaciones finales del efector antes de proceder a su implementación física. Para ello se creó el siguiente código el cual utiliza la Robotics Toolbox para crear y visualizar un brazo robótico junto con su entorno. Primero, se define el brazo robótico con sus características, y se muestra su configuración inicial. Luego, se calcula la cinemática directa para determinar la posición del efector final en función de las configuraciones articulares. Posteriormente, se definen puntos en el espacio tridimensional que el brazo robótico debe alcanzar, y se visualizan estos puntos con diferentes colores. Además, se representa una trayectoria de puntos intermedios entre la posición inicial y final que el robot seguirá. Este código es útil para planificar y verificar los movimientos del brazo robótico en un entorno tridimensional.

Código de Creación del Brazo Robótico - Robotics Toolbox:

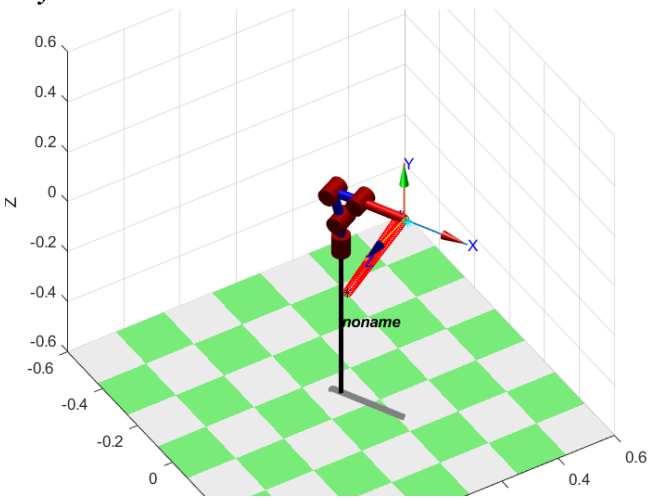
```
%Creacion matriz homogenea
L(1) = Revolute('d', 0.095, 'a', 0, 'alpha', pi/2);
L(2) = Revolute('d', 0, 'a', 0.105, 'alpha', 0);
L(3) = Revolute('d', 0, 'a', 0.098, 'alpha', 0);
L(4) = Revolute('d', 0, 'a', 0.155, 'alpha', 0);
R4D = SerialLink(L);
%Creacion robot
R4D.plot([0 0 0 0]);
%R4D.plot([25 10 -5 35]*pi/180);
%Cinematica Directa con splinecub1
tf = 5;
n = 50;
% [q1,w1,a1,t]=splinecub(0,0*pi/180,tf,n);
% [q2,w2,a2,t]=splinecub(0,120*pi/180,tf,n);
% [q3,w3,a3,t]=splinecub(0,-100*pi/180,tf,n);
% [q4,w4,a4,t]=splinecub(0,90*pi/180,tf,n);
R4D.plot([q1' q2' q3' q4']);
%R4D.plot([q1 q2 q3 q4]);

%Modelo cinematico directo
A= R4D.fkine([0 120 -100 90]*pi/180);


 %% Punto de Inicio
 % Define las coordenadas del punto en el espacio
x_punto = 0.2; % Cambia las coordenadas según tus necesidades
y_punto = -0.1; % Cambia las coordenadas según tus necesidades
z_punto = 0; % Cambia las coordenadas según tus necesidades
% Visualización del punto en el espacio
hold on
plot3(x_punto, y_punto, z_punto, '*k');

 %% Punto de Final - Cuadrado
 % Define las coordenadas del punto en el espacio
x_c = 0.18; % Cambia las coordenadas según tus necesidades
y_c = 0.1; % Cambia las coordenadas según tus necesidades
z_c = 0.2 ; % Cambia las coordenadas según tus necesidades
% Visualización del punto en el espacio
hold on
plot3(x_c, y_c, z_c, '*b'); % 'og' representa un punto verde

 %% Punto de Final - Rectangulo
 % Define las coordenadas del punto en el espacio
x_r = 0.2; % Cambia las coordenadas según tus necesidades
y_r = 0.1; % Cambia las coordenadas según tus necesidades
z_r = 0.2; % Cambia las coordenadas según tus necesidades
% Visualización del punto en el espacio
hold on
plot3(x_r, y_r, z_r, '*g'); % 'og' representa un punto verde

 %% Punto de Final - Circulo
 % Define las coordenadas del punto en el espacio
x_d = 0.22; % Cambia las coordenadas según tus necesidades
y_d = 0.1; % Cambia las coordenadas según tus necesidades
z_d = 0.2; % Cambia las coordenadas según tus necesidades
% Visualización del punto en el espacio
hold on
plot3(x_d, y_d, z_d, '*c'); % 'og' representa un punto verde

%% Mostrar Trayectoria
hold on
plot3(x_intermedias, y_intermedias, z_intermedias, 'or');
```

Al ejecutar los códigos anteriores en el siguiente orden: primero se ejecuta el código de la trayectoria de los tres tipos de clasificación, donde se especifica a través de la terminal de MATLAB el tipo de trayectoria. En este caso, se utilizan tres figuras geométricas (cuadrado, rectángulo y círculo), análogas a tres tipos de productos farmacéuticos. Luego, se procede a ejecutar el código de creación del brazo robótico, que utiliza la librería Robotics Toolbox para visualizar la construcción del robot, así como el movimiento traslacional y rotacional que realiza para llegar a su efector final establecido. Al llevar a cabo estas acciones, se obtienen estas tres trayectorias. Cabe destacar que en el punto negro es donde se recoge el producto para almacenar, y los otros tres puntos que se encuentran a una altura considerable hacen énfasis en la clasificación del producto. En las siguientes figuras se corrobora lo anterior.

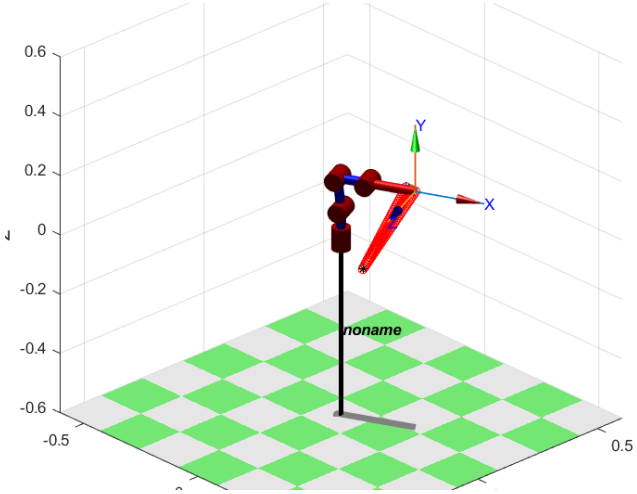
Trayectoria N°1



Trayectoria N°2



Trayectoria N°3



E. Ensamble del Brazo Robótico

Al realizar las diferentes etapas anteriores, se procedió con el ensamble en físico de los componentes para la construcción del brazo robótico de 4GDL. A continuación se presenta varias vistas:

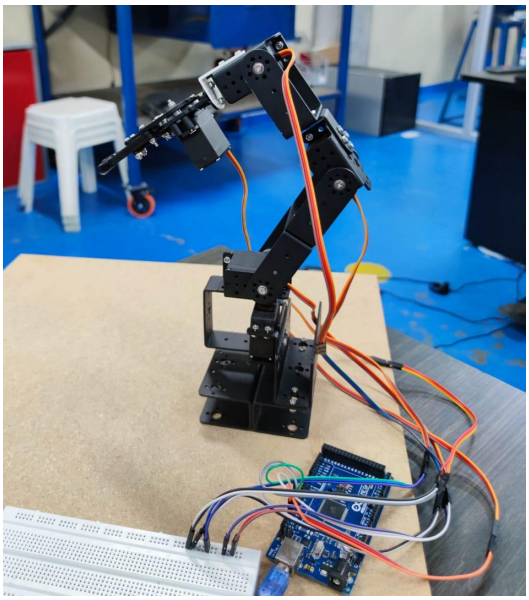


Figura 13. Brazo robótico 4GDL - Vista Lateral 1.

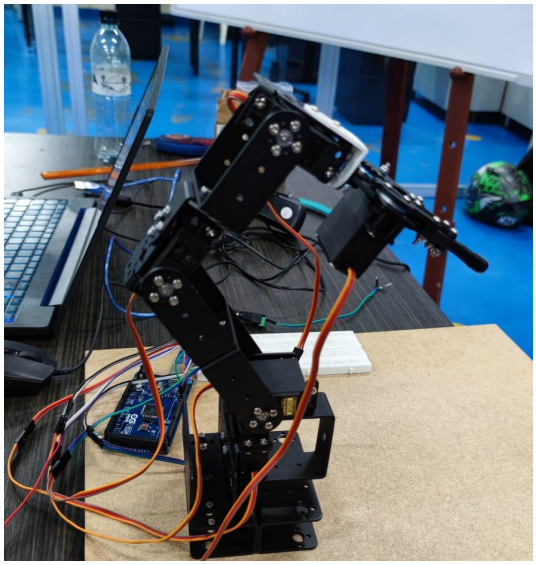


Figura 13. Brazo robótico 4GDL - Vista Lateral 2

F. Visión computarizada

La visión computarizada desempeña un papel esencial en la clasificación de figuras geométricas que representan diversos tipos de envases para productos farmacéuticos. En este contexto, OpenCV emerge como una herramienta fundamental para el procesamiento de imágenes. El proceso comienza con la adquisición de imágenes de envases farmacéuticos, seguido de pasos como la preprocesamiento, que incluye la eliminación de ruido y la mejora del contraste para optimizar la calidad de las imágenes. Luego, se procede a la detección de contornos, identificando las figuras geométricas en los envases. La extracción

de características clave de estos contornos, como la forma, el tamaño y la relación de aspecto, es un paso crucial para la posterior clasificación. OpenCV proporciona herramientas poderosas para llevar a cabo estos procedimientos de manera eficiente. Este enfoque ofrece una base sólida para la automatización de la clasificación de envases farmacéuticos, contribuyendo a la eficiencia y precisión en la industria farmacéutica.

Código:

```
import cv2
import numpy as np
import serial
import time
```

Se importan las bibliotecas necesarias para el procesamiento de imágenes (OpenCV y NumPy), la comunicación serial (serial), y el manejo del tiempo (time).

```
def detect_shape(frame, cnt):
    epsilon = 0.04 * cv2.arcLength(cnt, True)
    approx = cv2.approxPolyDP(cnt, epsilon, True)
    sides = len(approx)

    if sides == 3:
        return "TRIANGULO", b'1'
    elif sides == 4:
        x, y, w, h = cv2.boundingRect(approx)
        aspect_ratio = float(w) / h
        if abs(1 - aspect_ratio) < 0.05:
            return "CUADRADO", b'4'
        else:
            return "RECTANGULO", b'2'
    elif sides == 5:  # Check for a pentagon
        return "PENTAGONO", b'5'
    else:
        x, y, w, h = cv2.boundingRect(cnt)
        roi = frame[y:y+h, x:x+w]
        gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

        return "NINGUNA", b'0'
```

Esta función toma un contorno (cnt) y determina la forma geométrica basándose en el número de lados. Devuelve una etiqueta de forma y un valor asociado que se enviará al Arduino.

```
def main():
    arduino = serial.Serial("COM3", 9600)
    time.sleep(0)

    font = cv2.FONT_HERSHEY_COMPLEX

    cap = cv2.VideoCapture(1)

    last_send_time = time.time()  # Initialize the last send time

    while True:
        _, frame = cap.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        lower_green = np.array([60, 60, 60])
        upper_green = np.array([180, 255, 255])
        mask = cv2.inRange(hsv, lower_green, upper_green)
                        contours, _ = cv2.findContours(mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        figura, valor = "NINGUNA", b'0'
        for cnt in contours:
            area = cv2.contourArea(cnt)
            if area > 400:
                cv2.drawContours(frame, [cnt], 0, (0, 0, 255), 2)
                figura, valor = detect_shape(frame, cnt)

        cv2.putText(frame, "FORMA DETECTADA: ", (20, 30), font,
1, (255, 0, 0), 3)
        cv2.putText(frame, figura, (220, 70), font, 1, (0, 0, 255), 3)
        cv2.imshow("Original", frame)
        cv2.imshow("Mascara", mask)

        # Check if 10 seconds have passed since the last value was
sent
        if time.time() - last_send_time >= 5:
            arduino.write(valor)
            print(valor)
            last_send_time = time.time()  # Update the last send time

        key = cv2.waitKey(1)
        if key == 27:
            break

    cap.release()
    cv2.destroyAllWindows()

    try:
        arduino.close()
    except:
        pass

if __name__ == "__main__":
    main()
```

En la función principal (main), se realiza la inicialización de la comunicación serial, la configuración de la cámara, y un bucle principal que captura continuamente imágenes de la cámara, detecta formas, muestra la imagen con la forma detectada y envía información al Arduino cada 5 segundos.

G. Código de Arduino

```
#include <Servo.h>

// Definir pines para cada servo
int servoPin1 = 13;
int servoPin2 = 12;
int servoPin3 = 11;
int servoPin4 = 10;
int servoPin5 = 9;

Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;

bool figuraDetectada = false;  // Variable que indica si se detectó
una figura
bool trayectoriaEjecutada = false;  // Variable que indica si la
trayectoria ya se ejecutó
```

```
void setup() {
  Serial.begin(9600);

  // Inicializar los servos
  servo1.attach(servoPin1);
  servo2.attach(servoPin2);
  servo3.attach(servoPin3);
  servo4.attach(servoPin4);
  servo5.attach(servoPin5);

  // Posición Gripper
  mover_gripper(0);

  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);
}

void loop() {
  if (Serial.available() > 0) {
    byte data = Serial.read();  // Cambia char a byte
    Serial.print("Dato recibido: ");
    Serial.println(data);

    if (data == '1') {
      Serial.println("Encendiendo LED TRIANGULO");
      delay(1000);

      if (!figuraDetectada && !trayectoriaEjecutada) {
        ejecutar_trayectorias_2();
        figuraDetectada = true;  // Marcar que se detectó una figura
          trayectoriaEjecutada = true;  // Marcar que la trayectoria se
ejecutó
        delay(1000);
      }
    } else if (data == '5') {
      Serial.println("Encendiendo LED RECTANGULO");
      delay(1000);

      if (!figuraDetectada && !trayectoriaEjecutada) {
        ejecutar_trayectorias();
        figuraDetectada = true;  // Marcar que se detectó una figura
          trayectoriaEjecutada = true;  // Marcar que la trayectoria se
ejecutó
        delay(1000);
      }
    } else if (data == '2') {
      Serial.println("Encendiendo LED CIRCULO");
      delay(1000);

      if (!figuraDetectada && !trayectoriaEjecutada) {
        ejecutar_trayectorias_3();
        figuraDetectada = true;  // Marcar que se detectó una figura
          trayectoriaEjecutada = true;  // Marcar que la trayectoria se
ejecutó
        delay(1000);
      }
    } else {
        // Restablecer a la posición inicial si no se detecta ninguna
figura
      mover_a_posicion(80, 50, 120, 0, 180);
      figuraDetectada = false;  // Reiniciar la variable de detección
        trayectoriaEjecutada = false;  // Reiniciar la variable de la
trayectoria
      delay(1000);
    }
  }
}

void ejecutar_trayectorias() {
  // Posición Gripper
  mover_gripper(0);
  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);
  // Posición Secundario
  mover_a_posicion(80, 160, 150, 120, 180);
  // Posición Tercera
  mover_a_posicion(0, 160, 150, 120, 180);
  // Posición Agarrar
  mover_a_posicion(0, 90, 110, 180, 0);
  // Posición traslado objeto
  mover_a_posicion(0, 50, 110, 180, 180);
  // Subir Objeto
  mover_a_posicion(0, 160, 150, 120, 180);
  // Mover Objeto
  mover_a_posicion(180, 160, 150, 120, 180);
  // Dejar Objeto 1
  mover_a_posicion(180, 90, 110, 180, 180);
  // Dejar Objeto 2
  mover_a_posicion(180, 80, 110, 180, 180);
  // Dejar Objeto 3
  mover_a_posicion(180, 180, 110, 180, 0);
  //Posicición Inicial
  mover_a_posicion(80,160,150, 120,180);
  // Posición Gripper
  mover_gripper(180);
  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);

}

void ejecutar_trayectorias_2() {
  // Posición Gripper
  mover_gripper(0);
  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);
  // Posición Secundario
  mover_a_posicion(80, 160, 150, 120, 180);
  // Posición Tercera
  mover_a_posicion(0, 160, 150, 120, 180);

  // Posición Agarrar
  mover_a_posicion(0, 90, 110, 180, 0);
  // Posición traslado objeto
  mover_a_posicion(0, 50, 110, 180, 180);
  // Subir Objeto
  mover_a_posicion(0, 160, 150, 120, 180);
  // Mover Objeto
  mover_a_posicion(180, 160, 150, 120, 180);
  // Dejar Objeto 1
  mover_a_posicion(140, 60, 90, 180, 180);
  // Dejar Objeto 2
  mover_a_posicion(140, 90, 110, 180, 180);
  // Dejar Objeto 3
  mover_a_posicion(140, 90, 110, 180, 0);
  //Posicición Inicial
  mover_a_posicion(80,160,150, 120,180);
  // Posición Gripper
  mover_gripper(180);
```

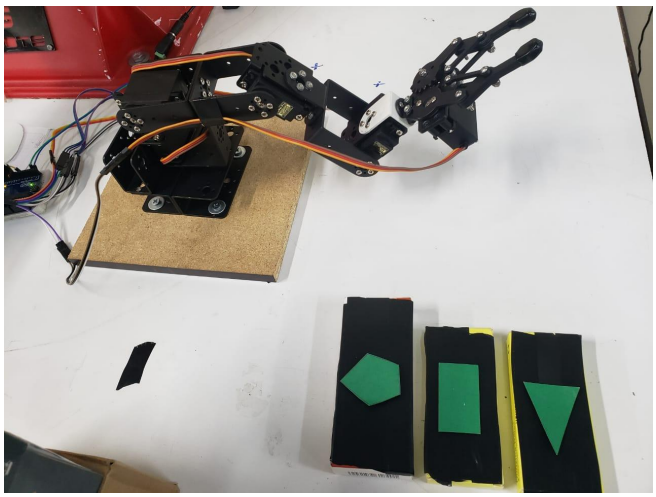
```
  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);
  trayectoriaEjecutada = false;
}


void ejecutar_trayectorias_3() {
  // Posición Gripper
  mover_gripper(0);
  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);
  // Posición Secundario
  mover_a_posicion(80, 160, 150, 120, 180);
  // Posición Tercera
  mover_a_posicion(0, 160, 150, 120, 180);
  // Posición Agarrar
  mover_a_posicion(0, 90, 110, 180, 0);
  // Posición traslado objeto
  mover_a_posicion(0, 50, 110, 180, 180);
  // Subir Objeto
  mover_a_posicion(0, 160, 150, 120, 180);
  // Mover Objeto
  mover_a_posicion(180, 160, 150, 120, 180);
  // Dejar Objeto 1
  mover_a_posicion(120, 60, 90, 180, 180);
  // Dejar Objeto 2
  mover_a_posicion(120, 80, 90, 180, 180);
  // Dejar Objeto 3
  mover_a_posicion(120, 80, 90, 180, 0);
  //Posicición Inicial
  mover_a_posicion(80,160,150, 120,180);
  // Posición Gripper
  mover_gripper(180);
  // Posición Inicial
  mover_a_posicion(80, 50, 120, 0, 180);
  trayectoriaEjecutada = false;
}

void mover_gripper(int angulo1) {
  servo4.write(angulo1);
  delay(500);  // Puedes ajustar este retardo según sea necesario
}

void mover_a_posicion(int angulo1, int angulo2, int angulo3, int
angulo4, int angulo5) {
  int paso = 1; // Define el incremento o decremento en cada paso
  int tiempo_entre_pasos = 10; // Define el tiempo entre cada paso

  int actual1 = servo1.read();
  int actual2 = servo2.read();
  int actual3 = servo3.read();
  int actual4 = servo4.read();
  int actual5 = servo5.read();

  // Interpolación lineal para suavizar el movimiento
  for (int i = 0; i <= 100; i += paso) {
    int pos1 = actual1 + i * (angulo1 - actual1) / 100;
    int pos2 = actual2 + i * (angulo2 - actual2) / 100;
    int pos3 = actual3 + i * (angulo3 - actual3) / 100;
    int pos4 = actual4 + i * (angulo4 - actual4) / 100;
    int pos5 = actual5 + i * (angulo5 - actual5) / 100;

    servo1.write(pos1);
    servo2.write(pos2);
    servo3.write(pos3);
    servo4.write(pos4);
    servo5.write(pos5);

    delay(tiempo_entre_pasos);
  }
}
```

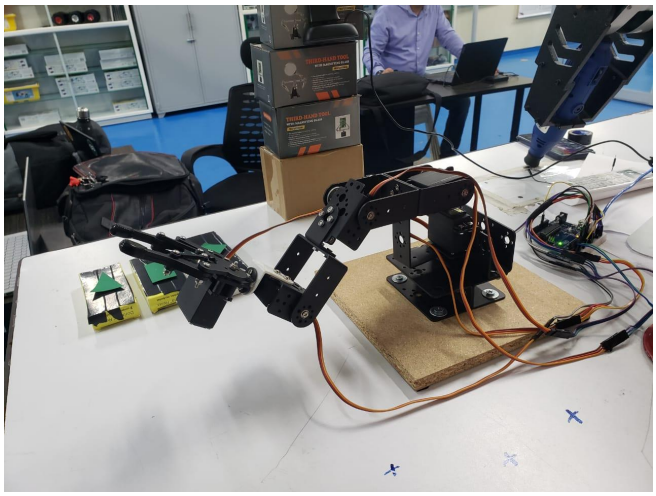
Este código en Arduino controla un brazo robótico con cinco servomotores para realizar movimientos y ejecutar trayectorias específicas en respuesta a señales enviadas por un programa en Python a través de comunicación serial. Aquí hay una descripción general del código:

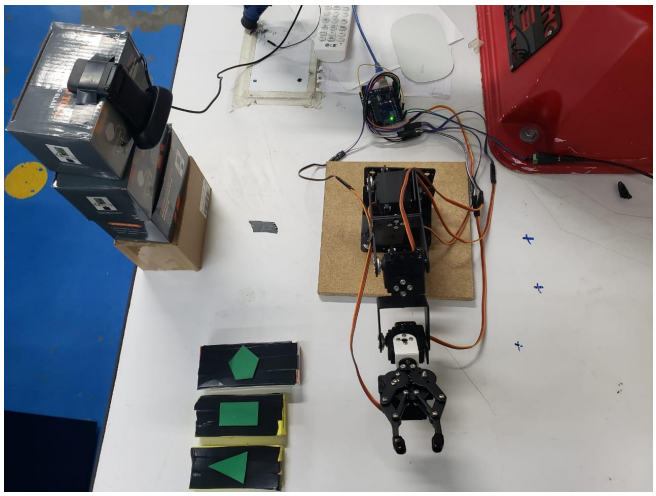
- *Definición de pines y objetos de servo*: Se definen los pines a los cuales están conectados los cinco servomotores, y se crean objetos de tipo Servo para cada uno de ellos.
- *Variables de control*: Se declaran dos variables booleanas, figuraDetectada y trayectoriaEjecutada, que indican si se ha detectado una figura y si la trayectoria asociada ya se ha ejecutado.
- *Función setup*: Se inicializan la comunicación serial y los objetos de servo. Además, se establece la posición inicial del gripper y del brazo.
- *Función loop*: En el bucle principal, se verifica si hay datos disponibles en el puerto serial. Cuando se recibe un dato, se identifica la figura asociada y se ejecuta la trayectoria correspondiente, encendiendo un LED virtual (simbolizado por mensajes en el puerto serial). Las trayectorias se ejecutan suavemente mediante la función mover_a_posicion. Si no se detecta ninguna figura, el brazo vuelve a la posición inicial.
- *Funciones de ejecución de trayectorias*: Se definen tres funciones (ejecutar_trayectorias, ejecutar_trayectorias_2, ejecutar_trayectorias_3) que describen las secuencias de movimientos del brazo para diferentes figuras (triángulo, rectángulo y círculo). Estas funciones utilizan la función mover_a_posicion para mover suavemente cada servo a las posiciones deseadas.
- *Funciones de movimiento y control*: mover_gripper: Controla la apertura/cierre del gripper (pinza).
- *mover_a_posicion*: Realiza una interpolación lineal para mover suavemente los servos a las posiciones deseadas.

IV.CONCLUSIONES

La implementación de un brazo robótico de 4 grados de libertad (4GDL) con visión artificial representa una innovación prometedora con el potencial de transformar la clasificación de medicamentos. A lo largo de este informe, se destacó la complejidad en su diseño y construcción, que involucra la integración de múltiples tecnologías, así como la importancia de la cinemática inversa en el control preciso del brazo robótico, de igual modo, la implementación de procesamiento de imagenes para vision artificial por medio del lenguaje de programación python utiilizando openCV con comunicación serial a arduino.







REFERENCIAS

[1] Craig, J. J. (2005). Introduction to Robotics: Mechanics and Control. Prentice Hall..

[2] Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). Robotics: Modelling, Planning and Control. Springer.

[3] Lynch, K. M., & Park, F. C. (2017). Modern Robotics: Mechanics, Planning, and Control. Cambridge University Press.

[4] Merlet, J. P. (2006). Parallel Robots. Springer.

[5] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. The International Journal of Robotics Research, 5(1), 90-98.

[6] Lin, J., & Hu, Y. (2012). A dynamic modeling method for 4-DOF parallel robots with multiple closed-loop subchains. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 289-294).

[7] Rahman, M. H., & Yusof, R. (2016). Dynamics and trajectory optimization of a 4-DOF redundant planar manipulator. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (pp. 1503-1508).

[8] Mavroidis, C., & Roth, B. (1992). Analysis of parallel manipulators with three and four degrees of freedom. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (pp. 358-363).