

# 3-Degree-of-Freedom Cylindrical Manipulator controlled by Computer Vision

Jordan Delgado Gutierrez

Faculty of Mechanical Engineering  
National University of Engineering  
Lima, Perú  
jordan.delgado.g@uni.pe

Carlos Aldave Torres

Faculty of Mechanical Engineering  
National University of Engineering  
Lima, Perú  
carlos.aldave.t@uni.pe

Bryan Gutierrez Fernandez

Faculty of Mechanical Engineering  
National University of Engineering  
Lima, Perú  
bryan.gutierrez.f@uni.pe

**Abstract**—This paper presents the design and implementation of a sign language translation glove that utilizes a sequential neural network for gesture prediction. The glove incorporates flex sensors to measure finger voltages and an accelerometer/gyroscope for orientation detection. It also includes a multiplexer, a 3.7V 450mAh battery with a 4056 charging module, and a flat micro vibration motor. The sensor inputs are processed using MicroPython on a Raspberry Pi Pico W, and the collected samples are used for training the neural network. The glove communicates with a web page using the MQTT protocol and Node-RED. The web page, developed using HTML, CSS, and JavaScript, displays the translated sign language gestures. The paper provides an overview of the problem, describes the methodology employed, presents the results obtained, and concludes with the key findings and future directions for improvement.

**Index Terms**—Sign language translation glove, sequential neural network, flex sensors, accelerometer/gyroscope, MQTT protocol, Node-RED, MicroPython, Raspberry Pi Pico W

## I. INTRODUCTION

Sign language serves as a crucial mode of communication for individuals with hearing impairments. However, the language barrier between sign language users and non-sign language users can hinder effective communication. Peruvian Sign Language (LSP) is the recognized sign language used by the Deaf community in Peru. In 2018, it gained official recognition as a national language, a significant step toward promoting inclusion and protecting the rights of Deaf individuals. Sign language interpreters are in high demand, facilitating communication between Deaf and hearing individuals across various settings. Efforts have been made to integrate sign language into the education system, ensuring equal access to education for Deaf students. Dedicated Deaf organizations tirelessly advocate for the rights and well-being of the Deaf community in Peru. It's worth noting that regional variations or dialects of LSP may exist, similar to spoken languages with regional accents or dialects. The media has also embraced the importance of sign language, incorporating interpretation services during television broadcasts and news programs. To address this issue, this paper proposes a sign language translation glove that translates hand gestures into text or speech, enabling better communication between sign language users and non-sign language users. The glove incorporates advanced sensing capabilities and employs a sequential neural network for accurate gesture prediction.

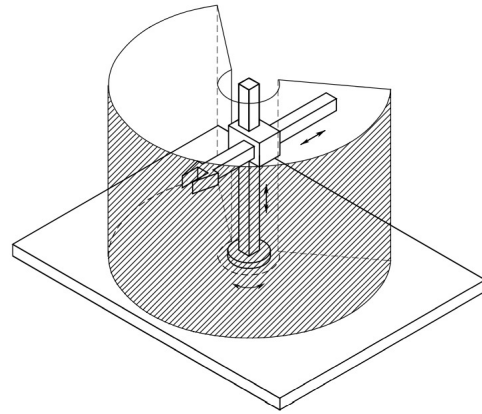


Fig. 1. Manipulador cilíndrico y su espacio de trabajo [3]

## II. PRESENTACIÓN DEL PROBLEMA

Aunque los brazos robóticos inteligentes pueden utilizarse en muchas industrias, el desarrollo de aplicaciones de brazos robóticos sigue siendo un reto importante. Un problema crucial y difícil es garantizar que la planificación de la trayectoria del brazo robótico sea precisa, segura y eficiente. La planificación de trayectorias consiste en conseguir que el efector final llegue al objetivo sin colisionar con obstáculos. Se basa en varios algoritmos que dictan el movimiento del brazo robótico y determinan cómo debe acercarse, procesar y orientarse para lograr una productividad óptima y evitar colisiones. Dado que el efector final de los brazos robóticos es fundamental para lograr su objetivo, llevar el efector final a la ubicación deseada evitando obstáculos es un reto clave para estos robots [4]. Además, muchas aplicaciones, como las utilizadas en el área médica, requieren una gran precisión para realizar correctamente sus funciones, precisión que es difícil de alcanzar con los métodos de posicionamiento tradicionales.

## III. DESCRIPCIÓN DEL MÉTODO A APLICAR

### A. Cinemática directa

a) *Configuración inicial*: Para la cinemática directa, se utilizaron programas de Matlab para manejar las ecuaciones de la cinemática directa. También se utilizó el IDE de Arduino,

que recibe el número de pasos que tienen que girar los motores paso a paso bipolares para moverse según los datos introducidos en la guía de Matlab. A continuación, se abre el puerto serie de Arduino en Matlab como se muestra en la figura 2. A continuación, se define el punto en el que se

```
%-----%
% A. ABRIMOS EL PUERTO SERIAL
%-----%
s1 = serial('COM10');
fopen(s1);
handles.puerto = s1; % Guardamos el puerto
disp('...abriendo puerto serial')
```

Fig. 2. Abrir el puerto serial de Arduino en Matlab

situará el robot al inicio del programa (posición inicial), tal y como se muestra en la siguiente figura 3. Las constantes del robot también se colocan en la lista DATA y los valores de las variables se almacenan en q1old, q2old y q3old para su posterior manejo. Las coordenadas del efector final en

```
%-----%
% B. VALORES INICIALES DEL ROBOT (HOME POSITION)
%-----%
% B.1. VALORES INICIALES DE LAS JUNTAS
q1 = 0; % th1 revolucion (sexag)
q2 = 150; % d2 prismatico 120mm
q3 = 0; % d3 prismatico 15mm
set(handles.q1, 'String',round(q1*1000)/1000);
set(handles.q2, 'String',round(q2*1000)/1000);
set(handles.q3, 'String',round(q3*1000)/1000);

% B.2. ESTRUCTURA PARA ALMACENAR LAS CONSTANTES DEL ROBOT
DATA.d1 = 15/100; % En metros |
DATA.a1 = 0;
DATA.a2 = 15/100;
DATA.a3 = 0;
handles.DATA = DATA;
% B.3. VARIABLES PARA ALMACENAR EL ULTIMO VALOR DE LAS JUNTAS
handles.q1_old = q1;
handles.q2_old = q2;
handles.q3_old = q3;
```

Fig. 3. Definición de la posición inicial

la posición inicial se calculan mediante las ecuaciones de la cinemática directa, como se muestra en la figura 4. A continuación, se trazan los sistemas de coordenadas de la posición inicial mediante el código que se muestra en la figura ??.

b) *Configuración del botón CINEMÁTICA:* Se toman los valores de cada articulación, que se introducen a través de los cuadros de texto de la guía de Matlab. Nótese que estos valores introducidos son cadenas de texto, por lo que deben ser convertidos a un valor numérico mediante la función str2double para poder trabajar con dichos datos. También se añaden las restricciones de movimiento de cada articulación, tal y como se muestra en la siguiente figura. De esta forma, se mostrará una ventana de error cada vez que se introduzca un valor fuera de rango para evitar problemas con el robot.

A continuación, se calculan las nuevas coordenadas a partir de los nuevos valores de cada articulación mediante

```
%-----%
% C. HALLAMOS LA CINEMATICA PARA LA CONF. INICIAL
%-----%
q1_rad = q1*pi/180; % A radianes
q2_metros = q2/1000; % A metros
q3_metros = q3/1000;
[T01,T02,T03] = cinematica_directa(q1_rad,q2_metros,...
    q3_metros, DATA);
x = T03(1,4);
y = T03(2,4);
z = T03(3,4);
set(handles.x, 'String',round(x*1000)/1000);
set(handles.y, 'String',round(y*1000)/1000);
set(handles.z, 'String',round(z*1000)/1000);
```

Fig. 4. Cálculo de las coordenadas de la posición inicial

```
%-----%
% CONFIGURACION DE LA FIGURA PARA EL ROBOT
%-----%
view(3)
grid on
axis([-0.5 0.5 -0.5 0.5 0 0.6])
xlabel('x(m)')
ylabel('y(m)')
% SISTEMA INERCIAL - TIERRA
T0 = eye(4);
plot_frame(T0,'frame','O', 'length', 0.1);
%-----%
% PLOTEAMOS LOS SISTEMAS COORDENADOS
h1 = plot_frame(T01,'frame','1', 'length', 0.1);
h2 = plot_frame(T02,'frame','2', 'length', 0.1);
h3 = plot_frame(T03,'frame','3', 'length', 0.1);
handles.h1 = h1;
handles.h2 = h2;
handles.h3 = h3;
%-----%
% PLOTEAMOS LOS ESLABONES
% ESLABON 1
h_link1 = plot_link(T01,T0,DATA.a1, 'k');
% ESLABON 2
h_link2 = plot_link(T02,T01,DATA.a2, 'k');
% ESLABON 3
h_link3 = plot_link(T03,T02,DATA.a3, 'm');
% GUARDAMOS LOS HANDLES
handles.h_link1 = h_link1;
handles.h_link2 = h_link2;
handles.h_link3 = h_link3;
```

Fig. 5. Trazado de la posición inicial

las ecuaciones de la cinemática directa, como se muestra en la siguiente figura 8 y se trazan los nuevos sistemas de coordenadas.

c) *Configuración del botón ENVIAR:* Mediante este botón se enviarán al arduino los valores de cada articulación que están escritos en la guía para que el robot realice el movimiento. El valor delta representa la variación de cada valor de articulación introducido con respecto al valor anterior, se calcula como se muestra en la figura 9. Los valores de transmisión también se ajustan en función de las características físicas del robot.

El número de pasos que debe dar cada motor se calcula como se indica en la figura 11. Tenga en cuenta que debe ser

```

% --- Executes on button press in boton_cinematica.
function boton_cinematica_Callback(hObject, eventdata, handles)
% hObject handle to boton_cinematica (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

%------%
% 1. LEEMOS EL VALOR DE LOS ANGULOS (caja de texto)
%------%
q1 = get(handles.q1, 'String'); % esta en sexag
q1 = str2double(q1);
q2 = get(handles.q2, 'String'); % esta en mm
q2 = str2double(q2);
q3 = get(handles.q3, 'String'); % esta en mm
q3 = str2double(q3);
disp('---')
fprintf('q1(sexag):%2.4f\n', q1)
fprintf('q2(mm):%2.4f\n', q2)
fprintf('q3(mm):%2.4f\n', q3)

```

Fig. 6. Lectura de datos

```

%------%
% 2. VERIFICAMOS LIMITACIONES
%------%
if(q1<0 || q1>180)
    errordlg('q1 fuera de rango (0,180)', 'q1')
    return
end
if(q2<10 || q2>285)
    errordlg('q2 fuera de rango (10mm,285mm)', 'q2')
    return
end
if(q3<0 || q3>140) % Prismático
    errordlg('q3 fuera de rango (0mm,140mm)', 'q3')
    return
end

```

Fig. 7. Limitaciones de las juntas

```

%------%
% 3. ACTUALIZAMOS LA CINEMATICA DIRECTA
%------%
q1_rad = q1*pi/180;
q2_metros = q2/1000;
q3_metros = q3/1000;
[T01,T02,T03] = cinematica_directa(q1_rad,q2_metros,...
    q3_metros, handles.DATA);
x = T03(1,4);
y = T03(2,4);
z = T03(3,4);
set(handles.x, 'String', round(x*1000)/1000);
set(handles.y, 'String', round(y*1000)/1000);
set(handles.z, 'String', round(z*1000)/1000);

```

Fig. 8. Envío de nuevas coordenadas al guide

```

%------%
% 3. HALLAMOS LOS DELTAS DE CADA JUNTA
%------%
% 3.1. PARA LA JUNTA 1 - MOTOR 1
q1_delta = q1 - handles.q1_old; % Esta en sexag
if(q1_delta>=0)
    %q1_delta = q1_delta;
    q1_dir = 1;
else
    q1_delta = -q1_delta;
    q1_dir = 0;
end
% 3.2. PARA LA JUNTA 2 - MOTOR 2
q2_delta = q2 - handles.q2_old; % En mm
if(q2_delta>=0)
    %q2_delta = q2_delta;
    q2_dir = 1;
else
    q2_delta = -q2_delta;
    q2_dir = 0;
end
% 3.3. PARA LA JUNTA 3 - MOTOR 3
q3_delta = q3 - handles.q3_old;
if(q3_delta>=0)
    %q3_delta = q3_delta;
    q3_dir = 1;
else
    q3_delta = -q3_delta;
    q3_dir = 0;
end

```

Fig. 9. Delta de cada junta

```

%------%
% 4. APLICAMOS LA TRANSMISION PARA HALLAR EL GIRO
% QUE DEBE REALIZAR LOS MOTORES
%------%
% FACTOR DE TRANSMISION DE LAS JUNTAS
k1 = 4.3;
k2 = 44.84;
k3 = 9;
% DELTA EN EL EJE DEL ROTOR
delta_motor1 = k1*q1_delta;
delta_motor2 = k2*q2_delta;
delta_motor3 = k3*q3_delta;

```

Fig. 10. Factores de transmisión de cada junta

un número entero, por lo que el valor obtenido se redondea. Finalmente, los datos se envían al arduino como se muestra en la figura 12 para que el robot realice el movimiento.

## B. Gripper

a) *Conceptos y librerías:* Para el movimiento del gripper desarrollamos un algoritmo de Computer Vision para el mapeo de manos (Hand Tracking), que nos permite extraer las coordenadas de 21 puntos de nuestra mano, tal y como se muestra en la figura 13. Para este mapeo se decidió emplear el lenguaje de programación python, debido a la gran cantidad de librerías que existen para el procesamiento de imágenes. Como se muestra en la figura 14, para este proyecto empleamos la librería “OpenCV” y el paquete “CVZone” que nos permiten trabajar con diversas herramientas de visión por computadora, tales como “HandDetector” que es la herramienta que empleamos para este proyecto.

Para la conexión entre arduino y python utilizaremos la librería “pyfirmata”. Esta librería nos permite utilizar el lenguaje

```

%-----%
% 5. HALLAMOS LOS PASOS Y DIRECCION PARA CADA MOTOR
%-----%
% 5.1. MOTOR 1
N1 = 200; % Pasos por vuelta de mi motor
m1_pasos = N1*delta_motor1/360;
m1_pasos = round(m1_pasos); % Convertimos a entero
m1_dir = q1_dir;
fprintf('m1_pasos:%d, m1_dir:%d\n', m1_pasos,m1_dir)
% 5.2. MOTOR 2
N2 = 200; % Pasos por vuelta de mi motor
m2_pasos = N2*delta_motor2/360;
m2_pasos = round(m2_pasos); % Convertimos a entero
m2_dir = q2_dir;
fprintf('m2_pasos:%d, m2_dir:%d\n', m2_pasos,m2_dir)
% 5.3. MOTOR 3
N3 = 200; % Pasos por vuelta de mi motor
m3_pasos = N3*delta_motor3/360;
m3_pasos = round(m3_pasos); % Convertimos a entero
m3_dir = q3_dir;
fprintf('m3_pasos:%d, m3_dir:%d\n', m3_pasos,m3_dir)

```

Fig. 11. Cálculo del paso del motor

```

%-----%
% 6. ENVIAMOS LOS ANGULOS AL ARDUINO
%-----%
% LO CONVERTIMOS EN UNA CADENA
cad = strcat(num2str(m1_pasos), ',', num2str(m1_dir), ...
            ',', num2str(m2_pasos), ',', num2str(m2_dir),...
            ',', num2str(m3_pasos), ',', num2str(m3_dir));
fprintf('cadena a enviar: %s\n', cad)
% ENVIAMOS AL ARDUINO
fprintf(handles.puerto, '%s', cad);

```

Fig. 12. Envío de datos a arduino

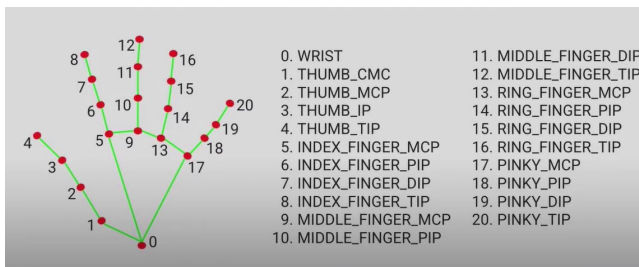


Fig. 13. Mapeo de los 21 puntos de nuestra mano



Fig. 14. Librerías usadas para visión por computador

de programación python para controlar la placa Arduino, tal como se muestra en la figura 3.15.

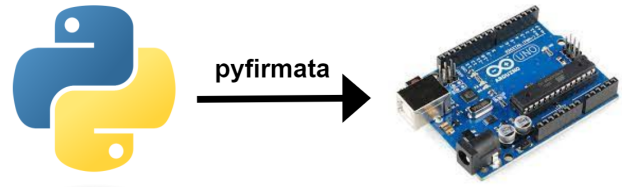


Fig. 15. Conexión Python - Arduino

b) *Explicación del código:* Primero guardamos la captura de la cámara incorporada en la laptop (0) en la variable “cap”, luego definimos las dimensiones de esta captura a 1280x720 píxeles. Como se muestra en la figura 16.

```

cap = cv2.VideoCapture(0)#leemos la cámara icorporada en la laptop
cap.set(3, 1280)#definimos el ancho de la captura como 1280 pixeles
cap.set(4, 720)#definimos el alto de la captura como 720 pixeles

```

Fig. 16. Lectura de cámara y dimensiones de la ventana de captura

Declaramos el detector de las manos con un threshold o confianza de detección de 80% y una cantidad máxima de 2 manos que puede detectar. Como se muestra en la figura 17.

```

detector = HandDetector(detectionCon=0.8, maxHands=2)

```

Fig. 17. Declaramos el detector de manos

Declaramos el puerto donde va conectado el arduino, en nuestro caso “COM5”, además de los pines de los 2 servos que emplearemos para el proyecto, en nuestro caso pin 5 para el servo de la pinza y pin 6 para el servo rotacional. Como se muestra en la figura 18.

```

#Declaramos los parámetros de la placa y los pines
port = "COM5"
board = pyfirmata.Arduino(port)
servo_pinX = board.get_pin('d:5:s') #pin 5 Arduino
servo_pinY = board.get_pin('d:6:s') #pin 6 Arduino

```

Fig. 18. Declaramos los parámetros de la placa y los pines

Declaramos las variables del programa, tales como la distancia mínima y máxima entre dedo índice(8) y pulgar (4), también el ángulo mínimo y máximo que puede girar el servo de la pinza. Por último declaramos la mínima y máxima distancia de la barra indicadora de la interfaz mostrada en la figura 12. Todas estas variables mencionadas se declaran como se muestra en la figura 19.

```
#Declaración de variables
minHand, maxHand = 20, 220
minDeg, maxDeg = 180, 90
minBar, maxBar = 400, 150
```

Fig. 19. Declaración de variables

Ejecutamos un bucle sin fin y guardamos en la variable “hands” las coordenadas de los puntos de las manos que encuentre en la imagen capturada. Como se muestra en la figura 8 20.

```
while True:
    success, img = cap.read()
    hands = detector.findHands()
    if hands:
```

Fig. 20. Función encontrar manos

Si detecta una mano izquierda entonces, guarda en la variable “lengthl” la distancia entre el punto 8 y 4 de la mano izquierda, es decir la distancia entre dedos índice y pulgar. De igual forma con la mano derecha guarda la distancia en la variable “lengthr”, como se muestra en la figura 21.

```
# Hand 1 (Left)
if hands:
    hand_l = hands[0]
    lmList_l = hand_l["lmList"] # List of 21 Landmark points
    length_l, info_l, img = detector.findDistance(lmList_l[8], lmList_l[4], img)

# Hand 2 (Right)
if len(hands) == 2:
    hand_r = hands[1]
    lmList_r = hand_r["lmList"] # List of 21 Landmark points
    length_r, info_r, img = detector.findDistance(lmList_r[8], lmList_r[4], img) # with draw
```

Fig. 21. Extracción de distancia entre dedos índice y pulgar

Realizamos una interpolación de la distancia calculada en la 21 para escalar la distancia a un ángulo que se encuentre en un rango que pueda girar el servo de la pinza, entre 180 (Cuando la pinza esté cerrada) y 90 ( Cuando la pinza esté abierta). Para el caso del servo rotacional lo colocamos en un rango de ángulo entre 0 y 180. Como se muestra en la figura 22.

Por último colocamos dos rectángulos en los extremos horizontales de la ventana de captura de video, que nos indiquen en todo momento la distancia de los dedos en un diagrama de barras que se mostrará en la figura 27. El código se muestra en la figura 23.

```
servoX = np.interp(length_l, [minHand, maxHand], [minDeg, maxDeg])
servoY = np.interp(length_r, [minHand, maxHand], [0, 180])
print(servoX)
print(servoY)
barX = np.interp(length_l, [minHand, maxHand], [minBar, maxBar])
barY = np.interp(length_r, [minHand, maxHand], [minBar, maxBar])
```

Fig. 22. Interpolación de distancia a ángulo

```
cv2.rectangle(img, (1180, 150), (1215, 400), (255, 0, 0), 3)
cv2.rectangle(img, (1180, int(barX)), (1215, 400), (0, 255, 0), cv2.FILLED)

cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 0), 3)
cv2.rectangle(img, (50, int(barY)), (85, 400), (0, 255, 0), cv2.FILLED)
```

Fig. 23. Diagrama de barras en tiempo real para las distancia de dedos

## IV. RESULTADOS

### A. Cinemática directa

Se desarrolló una interfaz de usuario como la que se muestra en la figura 24. Fijémonos en los valores de las articulaciones al inicio, éstos corresponden a la posición de Inicio, es decir, la posición en la que arranca el manipulador.

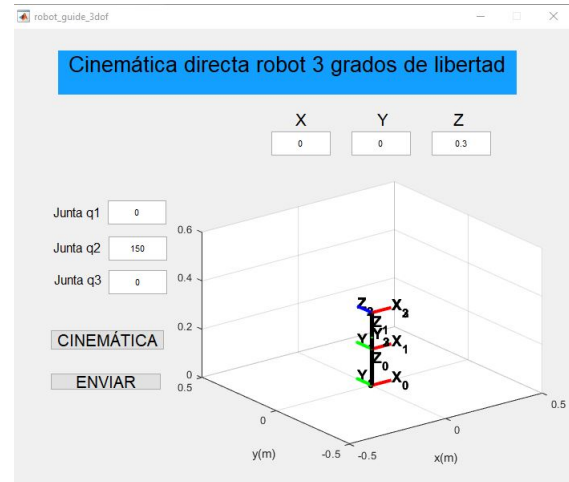


Fig. 24. Guía de posición inicial

Cuando se introduce un valor fuera de los rangos establecidos, se abre un cuadro de diálogo indicando que el valor conjunto está fuera de las normas. Cuando se pulsa el botón “ENVIAR” en el GUIDE, los datos introducidos se envían al arduino, y de esta forma, el código del arduino procesa estos valores para generar los pulsos que mueven los motores paso a paso según estos valores. Los valores de delta y número de pasos de cada articulación se muestran, como se ve en la figura 26.

### B. Gripper

Se desarrolló una interfaz amigable con dos diagramas de barras en tiempo real que grafican en todo momento el porcentaje de distancia entre los dedos pulgar e índice de ambas manos, izquierda y derecha. Además se realizó una

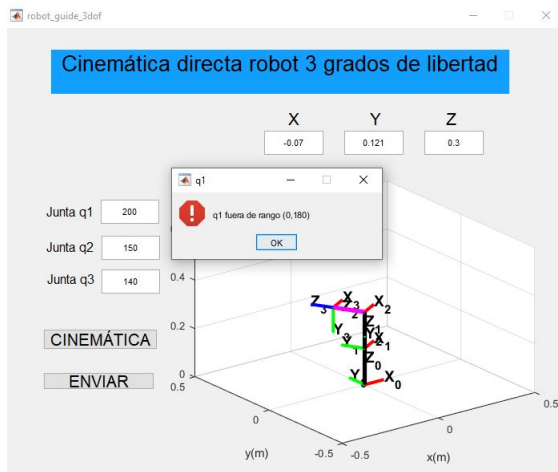


Fig. 25. Diálogo de error en la guía

```
q1 (sexag) :45.0000
q2 (mm) :150.0000
q3 (mm) :140.0000
q1_delta (sexag)=45.0000, q1_dir:1
q2_delta (mm)=0.0000, q2_dir:1
q3_delta (mm)=140.0000, q3_dir:1
m1_pasos:108, m1_dir:1
m2_pasos:0, m2_dir:1
m3_pasos:700, m3_dir:1
```

Fig. 26. Valores de las juntas enviados a arduino

interpolación de la distancia entre dedos para convertirlo a una escala de ángulo admitido por el servo, tanto de la pinza como el servo rotacional como se muestra en la figura 27.



Fig. 27. Interfaz final

## V. CONCLUSIONES

Finalmente, fue posible lograr un posicionamiento preciso para el Gripper con los tres grados de libertad del manipulador cilíndrico, a través de la amigable Interfaz de Usuario GUIDE de MATLAB, donde para la cinemática directa las entradas son ángulos y distancias y después de introducir estos valores primero el usuario puede apreciar una animación para saber cómo se moverá el manipulador y después de esto, puede confirmar este movimiento enviando estos valores.

Después de establecer la posición del manipulador, la pinza está lista para ser controlada por el movimiento de la mano del

usuario, quien podrá dirigir con precisión el comportamiento de la pinza, tanto su apertura como su rotación, y esto será imitado por la pinza en tiempo real.

En conclusión se ha conseguido diseñar un método de posicionamiento preciso para el efector final del manipulador cilíndrico de 3 grados de libertad y un seguimiento de la mano en tiempo real para permitir al usuario controlar de forma intuitiva y precisa el comportamiento de la pinza, pudiendo realizar cualquier tarea que el usuario decida completar sin mayores inconvenientes y con una alta fidelidad.

## REFERENCES

- [1] Dejan. (2 de 10 de 2020). How To Mechatronics. Obtenido de SCARA Robot — How To Build Your Own Arduino Based Robot: <https://howtomechatronics.com/projects/scara-robot-how-to-build-your-own-arduino-based-robot/>
- [2] Dermawan, R. (4 de 2 de 2022). GitHub. Obtenido de rizkydermawan1992/gesture-range-control: <https://github.com/rizkydermawan1992/gesture-range-control>
- [3] M. Spong, S. Hutchinson, and M. Vidyasagar. Robot Modeling and Control, 2nd Edition. Jhon Wiley Sons, 2020.
- [4] B. Siciliano, L. Sciacivco, L. Villani, and G. Oriolo. Robotics, Modelling, Planning and Control. Springer-Verlag, 2009.
- [5] B. Siciliano, and O. Khatib. Springer Handbook of Robotics, 2nd Edition. Springer-Verlag, 2016.
- [6] R. Murray, Z. Li, and S. Shankar. A Mathematical Introduction to Robotic Manipulation. CRC Press 1994.