



UNIVERSIDAD POLITÉCNICA DE JUVENTINO ROSAS

Proyecto Integrador

Replicación de movimientos con robot 3D y visión por computadora.

Carrera:

Ingeniería en Redes y Telecomunicaciones

Presentan:

Belman Casique Melissa

Obrajero Yañez Alma Lidia

Gutierrez Rodriguez Karla Berenice

Asesor:

Luis Rey Lara González

[Santa Cruz de Juventino Rosas, Gto. 3 de abril de 2025.](#)

Capítulo 1

Abstract

In this project, a 3D design of a humanoid robotic skeleton with integrated servo motors is used to mimic human movements. Although the replication is not completely accurate and there is a delay in the robot's response, the system is still able to mimic movements. The robot detects key points on the human body, such as wrists, elbows and shoulders, through a computer vision system.

Communication between the vision system and the robot's actuators is via Wi-Fi, which facilitates the transmission of data for the execution of movements, albeit with a slight delay in synchronization. In addition, an ESP32 module and an LCD display have been integrated to improve the interaction and control of the system.

Índice general

1. Abstract	2
2. Introducción	5
3. Objetivos	6
3.0.1. General	6
3.0.2. Específicos	6
3.0.3. Planteamiento del problema	7
3.0.4. Justificación	7
4. Marco teórico	8
5. Antecedentes	11
6. Desarrollo	13
6.1. Implementación de prototipo: esqueleto robótico	13
6.2. Interconexión entre Arduino (ESP32) y Python mediante Wi-Fi, Bluethooth	14
6.2.1. Importación de librerías en Python	14
6.3. Detección de rostros utilizando MediaPipe Face Detection	15
6.4. Ubicación de puntos clave de hombros, codos y muñecas	16
6.5. Control de servomotor con ESP32 y Python mediante visión por computadora	17
6.6. Control de movimiento en un esqueleto robótico con ESP32 y servomotores	17
6.7. Reconocimiento eficiente usando MediaPipe y OpenCV	18
6.8. Comunicación python-LCD128x64 ST7920	21
6.8.1. Código de reconocimiento facial	21
6.8.2. Demostración de reconocimiento de rostros	21
6.9. Seguimiento de persona con Arduino IDE	22
6.9.1. Lógica de Control	22
6.10. Reconocimiento de rostro con micropython	23
6.10.1. Instalación de Firmware MaixPy	24
6.10.2. Configuración de kflash-gui	24
6.11. Interface Maixpy	25

6.12. Creación de robot lina	25
7. Resultados	27
8. Conclusión	31

Capítulo 2

Introducción

La robótica hoy en día ha avanzado significativamente en las últimas décadas, permitiendo la creación de sistemas que imitan la anatomía y los movimientos humanos. Esta no solo tiene aplicaciones en la industria del entretenimiento y la educación, sino que también se extiende a extensos campos. En este contexto, el desarrollo de un esqueleto humanoide robótico que pueda emular los movimientos humanos se presenta como un desafío técnico y científico con una gran relevancia.

El presente proyecto se centra en el diseño y la implementación de un esqueleto humanoide robótico en 3D, equipado con servomotores que permiten la imitación de movimientos humanos. A pesar de que la replicación de estos movimientos no es completamente precisa y se enfrenta a un ligero retraso en la respuesta, el sistema demuestra una notable capacidad para emular la dinámica del cuerpo humano. Para lograr esta imitación, se ha integrado un sistema de visión por computadora que permite al robot detectar puntos clave del cuerpo, como muñecas, codos y hombros, facilitando así la sincronización de los movimientos.

La comunicación entre el sistema de visión y los actuadores del robot se realiza a través de una conexión Wi-Fi y Bluetooth, lo que permite una transmisión de datos eficiente, aunque con ciertas limitaciones en la sincronización. Además, se ha incorporado un módulo ESP32 y una pantalla LCD, que mejoran la interacción y el control del sistema, ofreciendo una interfaz más amigable para el usuario. Este proyecto no solo busca avanzar en la robótica humanoide, sino también explorar las posibilidades de interacción entre humanos y máquinas, abriendo nuevas vías para el desarrollo de tecnologías asistivas y de entretenimiento.

Capítulo 3

Objetivos

3.0.1. General

Desarrollar un robot que replique los movimientos humanos mediante visión por computadora y control de servomotores, ejecutando los movimientos de forma secuencial.

3.0.2. Específicos

- Comprender y aplicar las funcionalidades de MediaPipe para detectar y ubicar las articulaciones humanas, con el fin de facilitar la replicación de movimientos en el robot.
- Programar e integrar servomotores para que el robot replique los movimientos humanos con un control lo más preciso posible, considerando un ligero retraso en la ejecución.
- Implementar un algoritmo existente para interpretar los datos de movimiento de una persona y generar las señales necesarias para los servomotores, permitiendo una sincronización en tiempo casi real.

3.0.3. Planteamiento del problema

En la actualidad, la interacción entre humanos y robots se ha vuelto esencial en diversas áreas. Sin embargo, uno de los principales obstáculos que se enfrentan en la actualidad es la capacidad para imitar de manera precisa los movimientos humanos. A pesar de los avances en visión por computadora, la implementación de sistemas utilizan modelos de aprendizaje automático, como lo es MediaPipe, para la identificación de articulaciones.

Dentro de el entorno de MediaPipe la detección de articulaciones puede verse afectada por factores como la variabilidad en las posturas, las condiciones de iluminación y la complejidad de los movimientos, lo que puede resultar en una imitación inexacta. Además, el envío de las coordenadas de las articulaciones a comandos para los servomotores del robot requiere un algoritmo de control que asegure una sincronización adecuada, ya que sin un control de los movimientos del robot puede ser desincronizados, lo que afecta la calidad de la interacción entre humano y robot.

La capacidad de replicar movimientos en tiempo casi real es crucial para diversas aplicaciones, la latencia en el procesamiento de imágenes y la ejecución de comandos puede comprometer la fluidez de la interacción. Asimismo, un sistema que no pueda adaptarse a estas diferencias puede limitar la efectividad del robot en diversos entornos.

Este proyecto tiene como finalidad el desarrollar un sistema que utilice MediaPipe para la detección de articulaciones humanas, permitiendo que un robot replique dichos movimientos de manera precisa y en tiempo casi real. Se busca mejorar la interacción entre humanos y robots, facilitando una comunicación más natural y efectiva. Al abordar estos desafíos, se espera que el sistema contribuya a la integración de robots en la vida cotidiana, aumentando su funcionalidad y utilidad en diversas aplicaciones.

3.0.4. Justificación

Dada la creciente demanda de sistemas robóticos capaces de interactuar de manera más natural y eficiente con los seres humanos. La capacidad de replicar movimientos humanos tiene aplicaciones significativas en áreas como la asistencia robótica, la educación interactiva y el entretenimiento. Al utilizar visión por computadora para detectar puntos clave en la parte superior del cuerpo y sincronizar estos datos con un sistema de actuadores, se busca lograr una imitación casi precisa de los movimientos de los brazos humanos.

El uso de un esqueleto de robot impreso en 3D con espacios diseñados para colocar los servomotores es una parte clave del proyecto, lo que permite evaluar la eficacia de la comunicación e interactuar con el sistema desarrollado.

Además, el proyecto promueve el desarrollo de competencias en áreas clave de la ingeniería, como la visión por computadora, el control de movimientos y la integración de hardware y software.

Capítulo 4

Marco teórico

- **Servomotor SG90:** Un servomotor es un tipo especial de motor que permite controlar la posición del eje en un momento dado. Está diseñado para moverse determinada cantidad de grados y luego mantenerse fijo en una posición. En otras palabras, es un actuador rotativo o motor que permite un control preciso en términos de posición angular, aceleración y velocidad.



Figura 4.1: Microservomotor SG90.

- **Puente H:** Es un circuito electrónico que permite invertir la polaridad de tensión aplicada a una carga.



Figura 4.2: Puente H: L298N.

- **Motorreductor:** Es un sistema que combina un reductor de velocidad y un motor. Estos van unidos en una sola pieza y se usa para reducir la velocidad de un equipo de forma automática.



Figura 4.3: Motorreductor con caja reductora.

- **Pantalla LCD ST7920 gráfica (128×64):** es un tipo de pantalla de cristal líquido que consta de 128 columnas y 64 filas de píxeles, lo que da como resultado un total de 8192 píxeles. Estos píxeles se pueden controlar individualmente para mostrar texto, imágenes e incluso animaciones simples.

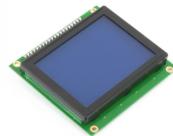


Figura 4.4: Pantalla gráfica LCD 128x64.

- **ESP32:** es un microcontrolador desarrollado por Espressif Systems que integra conectividad Wi-Fi y Bluetooth en un solo chip. Equipado con un procesador de doble núcleo de 32 bits, el ESP32 es altamente eficiente en términos de consumo de energía y rendimiento. Además, cuenta con uno conector micro USB para comunicación y alimentación.



Figura 4.5: Placa de desarrollo ESP32.

- **Fuente de alimentación 9v:** se utilizó una batería externa para suministrar la energía necesaria a todos los componentes del robot.

- **SIPEED MaixDuino:** es una placa de desarrollo que utiliza el módulo M1AI como unidad central. Este módulo incorpora un procesador de doble núcleo de 64 bits y 8 MB de SRAM. Además, está equipada con un módulo ESP32 (Wi-Fi+Bluetooth integrado), que se conecta fácilmente a internet.

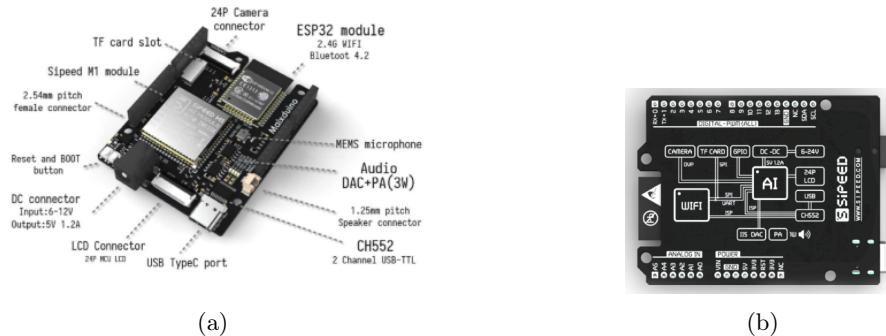


Figura 4.6: Placa de desarrollo MaixDuino.

Capítulo 5

Antecedentes

El presente proyecto ha tenido avances significativos con el pasar del tiempo. En cuatrimestres anteriores, se han abordado áreas fundamentales de la ingeniería, tales como electrónica, programación y visión por computadora. Se trabajó con un RoboSapien, un robot humanoide que posee la capacidad de recibir y ejecutar comandos mediante señales infrarrojas (IR). Este dispositivo fue seleccionado debido a su accesibilidad en la interpretación de señales IR y su estructura mecánica adaptable para la replicación de movimientos humanos.



(a)

(b)

Figura 5.1: Señales IF recibidas por el RoboSapien.

Para lograr una sincronización de los movimientos, se utilizó MediaPipe, una solución de visión por computadora desarrollada por Google, que permite el rastreo del esqueleto humano mediante la detección de puntos clave en tiempo real. Estos puntos clave representan las articulaciones principales del cuerpo, como hombros, codos y muñecas, lo que posibilita un análisis detallado de la postura de una persona.

Con el propósito de replicar estos movimientos en el RoboSapien, se desarrolló un código en Arduino que controla servomotores asociados a cada uno de los puntos clave detectados. Se estableció una relación directa entre los puntos clave del esqueleto y las articulaciones, logrando así una sincronización. Para ello, se realizaron cálculos de ángulos y desplazamientos que fueron transmitidos a los servomotores, permitiendo que el robot imitara los movimientos humanos.

observados. La comunicación entre el sistema de visión por computadora y el RoboSapien se llevó a cabo mediante señales infrarrojas, aprovechando el receptor IR integrado en el robot. Se diseñó un protocolo de comunicación IR que traduce los datos de movimiento en comandos específicos, los cuales son interpretados por el RoboSapien para ejecutar los movimientos detectados.

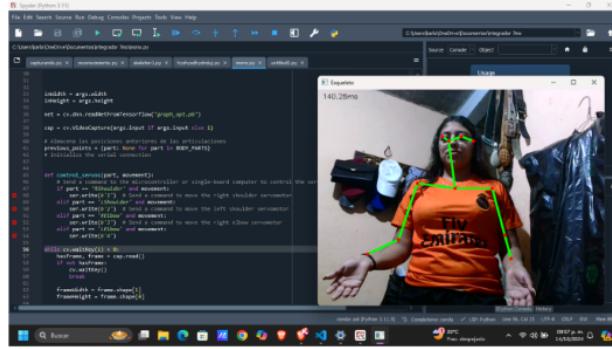


Figura 5.2: Demostración de skeleton.

Adicionalmente, para mejorar la experiencia de usuario e incrementar la interactividad del sistema, se incorporó una pantalla LCD. Esta pantalla muestra el nombre de la persona detectada, utilizando técnicas de reconocimiento facial combinadas con Openpose. La integración de la LCD no solo proporcionó información en tiempo real, sino que también aportó un valor añadido en términos de identificación y personalización de la interacción con el robot.



Figura 5.3: Mensaje mostrado en LCD.

Capítulo 6

Desarrollo

6.1. Implementación de prototipo: esqueleto robótico

En el desarrollo del proyecto, se empleó un prototipo de esqueleto robótico impreso en 3D, diseñado para la integración de servomotores en sus articulaciones. Además, se implementó visión por computadora para la detección y análisis de los movimientos humanos, permitiendo la replicación de estos en el robot mediante el control de los actuadores(servomotores).

En la figura 6.1 se muestra la estructura del esqueleto robótico y todas las piezas que lo conforman.

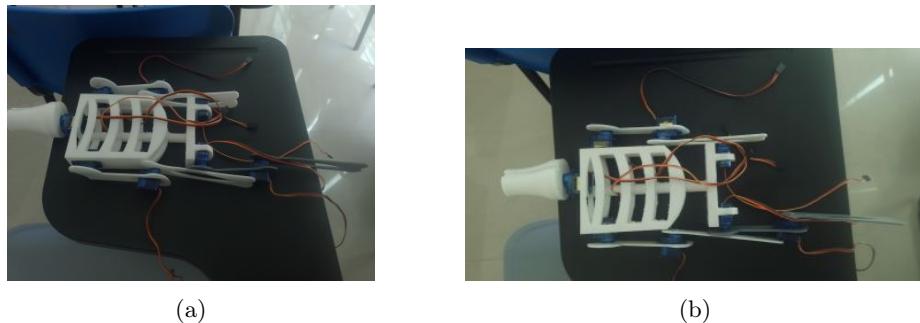


Figura 6.1: Montaje y ensamblaje de las piezas del robot.

6.2. Interconexión entre Arduino (ESP32) y Python mediante Wi-Fi, Bluethooth

6.2.1. Importación de librerías en Python

Para el procesamiento de imágenes y videos en tiempo real, se instalaron y configuraron las siguientes herramientas en el entorno de Python:

- **OpenCV:** Se integró la biblioteca OpenCV para disponer de la herramienta cv2, que permite el análisis y procesamiento eficiente de imágenes y videos en tiempo real.
- **MediaPipe:** Se implementó MediaPipe para facilitar el análisis de secuencias de video en tiempo real. Esta herramienta proporciona funciones avanzadas como el reconocimiento de gestos, el seguimiento facial y corporal, así como la detección de objetos, lo que optimiza la interpretación de movimientos.
- **Time:** Se usa para calcular la velocidad de movimiento del rostro detectado.

Además, para establecer la comunicación Wi-Fi entre Arduino (ESP32) y Python, se utilizó el protocolo de sockets, que permite el intercambio de datos a través de una red.

En Python, los sockets se implementan en el módulo socket. Para crear un socket en Python se siguieron los siguientes pasos:

1. Importar el módulo socket: `import socket`
2. Crear un objeto socket:
`sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
Donde: el primer argumento indica que se usará el protocolo IP versión 4 y el segundo argumento indica que se utiliza el protocolo de transporte TCP.
3. Conectar el socket con la ESP32:
`sock.connect((dirección IP, Puerto))`
En este caso se especificó la dirección ip de la ESP32 y el puerto 80.
4. Para enviar datos con el socket se programó la siguiente línea de código:
`sock.sendall(dato.encode())`
5. Finalmente, se cierra el socket: `sock.close()`

Para este caso el socket actúa como cliente, iniciando la conexión con la ESP32. Es decir, establece ese canal de comunicación, vinculándose a una dirección IP y a un puerto específico.

En la figura 6.2 se muestra el uso de sockets en Python para establecer comunicación con la ESP32.

```
import cv2
import mediapipe as mp
import time
import socket # Para comunicación WiFi

#----- Comunicación WiFi -----
esp32_ip = "192.168.1.71" # IP de la ESP32
esp32_port = 80

def conectar_esp32():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock.connect((esp32_ip, esp32_port))
        print("Conectado a ESP32")
        return sock
    except Exception as e:
        print(f"Error de conexión: {e}")
        return None
    sock = conectar_esp32()
```

Figura 6.2: Implementación de sockets en Python.

6.3. Detección de rostros utilizando MediaPipe Face Detection

En el caso de la detección de rostros, esto implica preprocesar los datos de entrada y detectar rostros en secuencias de video, para lograr este objetivo se uso la herramienta **Face Detection** la cual permite localizar la posición de los rostros e identificar puntos de referencia o puntos clave como la ubicación de los ojos, la nariz y la boca. Se utilizó el módulo integrado **Drawing Utils** de MediaPipe el cual proporciona funciones útiles para dibujar sobre frames de video. Para este proyecto, ayuda a visualizar los resultados del análisis de visión por computadora, como los puntos clave del cuerpo humano o la cara, facilitando la interpretación de los datos de manera gráfica.

Posteriormente, se procede a realizar la video captura de la cámara principal (ordenador), es importante mencionar que para garantizar que realmente se está detectando un rostro en sí, se declaró una variable llamada **rostros**, dicha variable almacena el umbral mínimo de 0.85 que arroja Face Detection, si esto es cierto, se comienza a guardar los fotogramas del vídeo.

```
# Establecer que detector tenga un umbral mínimo de 0.85 y asignar esto a rostros
with detector.FaceDetection(min_detection_confidence = 0.85) as rostros:
    while True:
        #Almacenar los fotogramas del video
        ret, frame = cap.read()
        if not ret:
            continue

        #Se mueve a la misma dirección
        frame = cv2.flip(frame, 1)

        rgb= cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        resultado = rostros.process(rgb)
```

Figura 6.3: Almacenamiento de los fotogramas del video.

Se utilizo la función `cv2.flip` para invertir de manera vertical el video, es decir, se aplico un efecto espejo, donde dicha función recibe los fotogramas y un `flipCode`(que especificará como voltear la imagen). A continuación, se convierte el fotograma a formato RGB (MediaPipe usa RGB en lugar de BGR).

En la figura 6.3 se muestra el código desarrollado referente a lo mencionado anteriormente.

Es muy importante diferenciar cada rostro detectado, en base a esto, se enumeró con un `id` cada resultado y se obtuvo la dimensión (ancho y alto) del frame con la herramienta `frame.shape`. Uno de los pasos muy importantes en el desarrollo del código fue extraer las coordenadas relativas del rostro y convertirlas en píxeles, dado que las coordenadas relativas mostradas eran números decimales y los píxeles suelen representarse con números enteros.

Finalmente, se programan distintas condiciones en varias sentencias `if` para enviar datos a la ESP32 acorde a la posición detectada.

6.4. Ubicación de puntos clave de hombros, codos y muñecas

Se creó una función llamada `calcular_angulo` la cual recibe tres puntos en coordenadas (x, y) y calcula el ángulo entre los segmentos formados por esos puntos. Para que los servomotores interpreten correctamente los datos recibidos por Python es necesario convertir el ángulo de radianes a grados implementando la función `math.degrees()`. A continuación, se extraen y escalan las coordenadas de los puntos clave del modelo de MediaPipe en este caso de hombros, codos, muñecas y cadera para posteriormente obtener el ángulo de hombros y codos mediante la función previamente declarada.

Es fundamental asegurarse de que los ángulos enviados a la ESP32 estén dentro del rango de 0° a 180°, para esto como paso final se utilizó la función `bytarray`, esta devuelve una nueva matriz de bytes llamada `angles`(ángulos).

Al final, se envían los datos por socket: `sock.sendall(angles)`.

En la figura 6.4 se muestra el método para calcular los ángulos de los hombros y codos derechos e izquierdos.

```
# calcular ángulos
elbow_angle_r = calcular_angulo(points["shoulder_r"], points["elbow_r"], points["wrist_r"])
shoulder_angle_r = calcular_angulo(points["hip_r"], points["shoulder_r"], points["elbow_r"])

elbow_angle_l = calcular_angulo(points["shoulder_l"], points["elbow_l"], points["wrist_l"])
shoulder_angle_l = calcular_angulo(points["hip_l"], points["shoulder_l"], points["elbow_l"])

# Limitar ángulos a 0-180
angles = bytarray([
    min(max(shoulder_angle_r, 0), 180),
    min(max(elbow_angle_r, 0), 180),
    min(max(shoulder_angle_l, 0), 180),
    min(max(elbow_angle_l, 0), 180)
])
```

Figura 6.4: Función calcular ángulo.

6.5. Control de servomotor con ESP32 y Python mediante visión por computadora

Para garantizar el correcto funcionamiento de los servomotores, es fundamental establecer una comunicación inalámbrica eficiente y estable entre Python y la ESP32. En este contexto, dentro del entorno de desarrollo Arduino IDE, se configuró la conexión Wi-Fi mediante la declaración del **SSID**(nombre de la red) y su respectiva contraseña. Además, se estableció el uso del **puerto 80**, permitiendo la sincronización y el intercambio de datos con Python.

Una vez establecida la comunicación, se configuraron los pines correspondientes para el control de los servomotores. En el proceso de inicialización, se definió un objeto de tipo Servo y se asignó un pin específico para su conexión. Además, se estableció un valor inicial de 90 grados para garantizar una posición en el arranque del sistema. Durante la ejecución del programa, se recibe información en formato de cadena a través de la comunicación establecida. Dependiendo de los datos recibidos, se ajusta el ángulo del servomotor en función de una escala mapeada, la cual traduce valores de velocidad en incrementos o decrementos del movimiento. Finalmente, el ángulo se limita dentro del rango permitido de 0 a 180 grados, asegurando un control preciso del servomotor.

6.6. Control de movimiento en un esqueleto robótico con ESP32 y servomotores

El código desarrollado permite el control de cuatro servomotores que representan los movimientos de los brazos del esqueleto de un robot. Se definen dos servomotores para cada brazo: hombro y codo, tanto para el derecho como para el izquierdo. En la fase de configuración `setup()`, se asignan los pines correspondientes a cada servomotor para establecer su conexión con la ESP32.

Durante la ejecución `loop()`, el programa espera la llegada de un cliente que envíe datos. Cuando un cliente se conecta, el código mantiene abierta la comunicación y espera recibir un paquete de cuatro valores representando los ángulos de los servomotores. Estos valores, que llegan en un rango de 0 a 255, se convierten a un rango de 0 a 180 grados utilizando la función `map()`. Posteriormente, los valores convertidos se envían a los servomotores para que adopten la posición indicada.

Finalmente, el código imprime en el monitor serie los ángulos asignados a cada servomotor, lo que permite monitorear el comportamiento del sistema en tiempo real. Además, se implementa un breve retraso de 50 ms para evitar una sobrecarga en el bucle de ejecución.

6.7. Reconocimiento eficiente usando MediaPipe y OpenCV

La visión por computadora es un campo que permite a los equipos informáticos interpretar y comprender su entorno. A través del uso de algoritmos avanzados y técnicas de aprendizaje automático, estos sistemas son capaces de analizar imágenes y videos para llevar a cabo tareas complejas, como la detección de rostros y cuerpos humanos. En el desarrollo de este proyecto, se empleó la biblioteca OpenCV debido a su eficiencia en el procesamiento de video.

La función `cv2.VideoCapture()` es esencial para capturar un vídeo desde una cámara, basta con crear un objeto que contenga dicha función y pasar como argumento el índice de cámara (0 para la cámara predeterminada).

```
# iniciar la cámara
cap = cv2.VideoCapture(0)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
```

(a) Lectura de cada fotograma utilizando `cap.read()`.

```
cv2.imshow("Detección", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

(b) Se utiliza `cv2.imshow` para mostrar cada fotograma en una ventana.

Figura 6.5: Procesamiento de video con `cv2`.

En la figura 6.5 se muestra el código para capturar el vídeo de la cámara predeterminada y mostrarlo en una ventana.

La estimación de la postura humana es el proceso de detección y seguimiento de puntos de referencia del cuerpo humano en imágenes y vídeos, para esta técnica de visión artificial se utilizó **MediaPipe**. Los puntos clave detectados suelen incluir articulaciones como codos, rodillas, hombros y tobillos.

MediaPipe es un marco de aprendizaje automático multiplataforma y de código abierto. Uno de sus modelos, MediaPipe Stance, es capaz de identificar hasta 33 puntos de referencia en 3D del cuerpo humano, como la cabeza, los hombros, los codos, las muñecas, las manos y las caderas, lo que permite estimar con precisión la postura de una persona.

En la figura 6.6 se muestra uno de los modelos implementados, previamente entrenados de Mediapipe.

A continuación, se describen los pasos para la detección de puntos de referencia utilizando MediaPipe:

1. **Inicialización de MediaPipe Pose:** Se importa y se inicializa el modelo `mp_pose.Pose()` de MediaPipe para detectar los puntos de referencia del cuerpo humano.

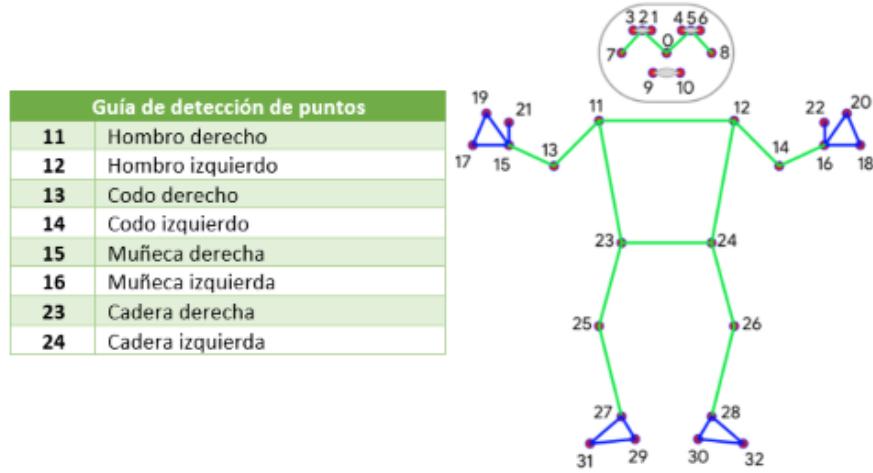


Figura 6.6: Puntos de referencia 3D predefinidos para la estimación de la postura humana.

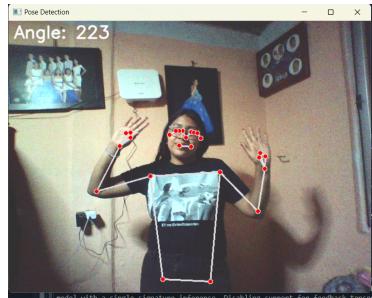


Figura 6.7: Modelo con puntos de referencia aplicados.

2. **Detección de puntos de referencia (landmarks):** El marco de video se convierte a formato RGB (`cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)`) para que pueda ser procesado por MediaPipe.
3. **Comprobación de detección de cuerpo:** Si `results.pose_landmarks` contiene datos, significa que MediaPipe ha detectado al menos un cuerpo humano.
4. **Conexiones de la parte superior del cuerpo:** Para cada par de puntos, se obtienen sus coordenadas y se traza una línea entre ellos utilizando `cv2.line()`, visualizando así las conexiones entre esos puntos del cuerpo.
5. **Dibujo de puntos de referencia:** Para cada punto de referencia en

```
# convierte la imagen a RGB para MediaPipe
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = pose.process(rgb_frame)
```

Figura 6.8: Conversión del marco de video a formato RGB.

```
# Dibuja la línea entre los puntos
start_point = int(start.x * frame.shape[1]), int(start.y * frame.shape[0])
end_point = int(end.x * frame.shape[1]), int(end.y * frame.shape[0])
cv2.line(frame, start_point, end_point, line_color, 2)
```

Figura 6.9: Conexiones entre puntos del cuerpo superior visualizadas con líneas.

la lista `upper_body_landmarks`, se extraen las coordenadas (x, y) y se dibujan en la imagen usando `cv2.circle()`.

```
landmarks = results.pose_landmarks.landmark
neck = (landmarks[8].x, landmarks[8].y)
left_shoulder = (landmarks[11].x, landmarks[11].y)
left_elbow = (landmarks[13].x, landmarks[13].y)
left_wrist = (landmarks[15].x, landmarks[15].y)
right_shoulder = (landmarks[12].x, landmarks[12].y)
right_elbow = (landmarks[14].x, landmarks[14].y)
right_wrist = (landmarks[16].x, landmarks[16].y)
```

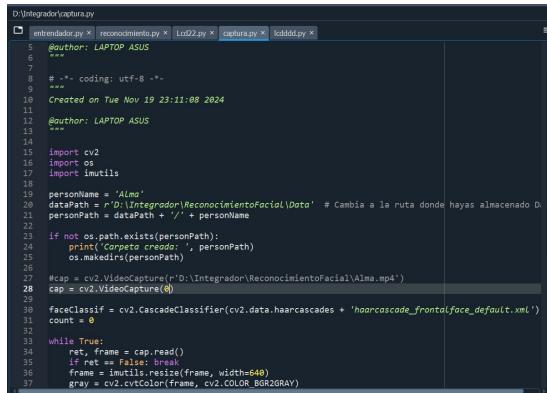
Figura 6.10: Valores predeterminados para puntos de referencia.

6.8. Comunicación python-LCD128x64 ST7920

Python en un entorno utilizado con Spyder que puede enviar datos a un microcontrolador a través de comunicación serial. Este microcontrolador procesa esos datos y los utiliza para mostrar información en una pantalla LCD ST7920. SPI es el protocolo utilizado entre el microcontrolador y la pantalla para la comunicación rápida y eficiente.

6.8.1. Código de reconocimiento facial

Este proyecto combina reconocimiento facial con visualización en pantalla LCD, así como Python con OpenCV este se encarga de realizar el reconocimiento facial y Python envía el nombre de la persona reconocida al microcontrolador (ESP32) a través de comunicación serial donde muestra el nombre en la pantalla LCD.



The screenshot shows a Python code editor in Spyder. The code is for facial recognition and is as follows:

```
 5  #!/usr/bin/python
 6  # Author: LAPTOP ASUS
 7  # www
 8  # -*- coding: utf-8 -*-
 9  # Created on Tue Nov 19 23:11:08 2024
10  # Author: LAPTOP ASUS
11  #
12  import cv2
13  import os
14  import imutils
15
16  personName = 'Alma'
17  dataPath = 'D:\Integrador\ReconocimientoFacial\Datos' # Cambia a la ruta donde hayas almacenado D.
18  personPath = dataPath + '/' + personName
19
20  if not os.path.exists(personPath):
21      print('Carpeta creada: ', personPath)
22      os.makedirs(personPath)
23
24  #cap = cv2.VideoCapture('D:\Integrador\ReconocimientoFacial\Alma.mp4')
25  cap = cv2.VideoCapture(0)
26
27  faceClassif = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
28  count = 0
29
30  while True:
31      ret, frame = cap.read()
32      if ret == True:
33          frame = imutils.resize(frame, width=640)
34          gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
35
36
37
```

Figura 6.11: Código de reconocimiento facial.

6.8.2. Demostración de reconocimiento de rostros

Se muestra un mensaje de la persona que está detectando en el algoritmo diseñado en python y este mismo se visualiza en la cara del robot para que muestre el nombre entre otros diseños.

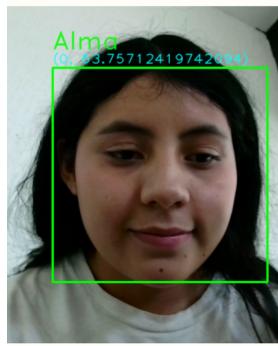


Figura 6.12: Persona detectada.

6.9. Seguimiento de persona con Arduino IDE

Este código está diseñado para controlar un robot móvil que sigue a una persona utilizando un sensor ultrasónico y sensores infrarrojos. El robot ajusta su dirección de movimiento en función de la posición de la persona, girando hacia donde ella se mueve.

NewPing: Es una biblioteca que facilita la interacción con el sensor ultrasónico, que mide la distancia a un objeto.

6.9.1. Lógica de Control

Lectura de Sensores Infrarrojos: Se leen los valores de los sensores infrarrojos izquierdo y derecho.

```
if (distance < DESIRED_DISTANCE - 5) { // Si el
| | stopMotors(); // Detiene los motores
} else if (distance > DESIRED_DISTANCE + 5) {
| | moveForward(); // Mueve hacia adelante
```

Figura 6.13: Medición de distancia.

- Si el objeto está demasiado cerca (menos de 15 cm), se detienen los motores.
- Si el objeto está demasiado lejos (más de 25 cm), el robot avanza hacia adelante.
- Si la distancia es adecuada (entre 15 cm y 25 cm), se verifica la información de los sensores infrarrojos para mantenerse pendientes.

```

    | | | // Si la distancia es adecuada, verifica los sensores IR
    | | if (leftSensor == HIGH && rightSensor == LOW) {
    | | | turnLeft(); // Gira a la izquierda si el sensor izquierdo detecta a
    | | } else if (rightSensor == HIGH && leftSensor == LOW) {
    | | | turnRight(); // Gira a la derecha si el sensor derecho detecta a la
    | | } else if (leftSensor == HIGH && rightSensor == HIGH) {
    | | | // Si ambos sensores detectan a la persona, el robot puede avanzar
    | | | moveForward(); // Avanza hacia la persona
    | | } else {
    | | | moveForward(); // Si no hay detección, avanza
    | }
}

```

Figura 6.14: Lógica de seguimiento.

Medición de Distancia: Se mide la distancia a la persona utilizando el sensor ultrasónico y se imprime en el monitor serial.

```

unsigned int distance = sonar.ping_cm()
Serial.print("Distance: ");
Serial.println(distance); // Imprime la

```

Figura 6.15: Medición en cm.

6.10. Reconocimiento de rostro con micropython

En esta sección se utiliza un nuevo ide MaixPy es un firmware de MicroPython diseñado para placas de desarrollo como la Sipeed Maix Bit y Maix-CAM. Proporciona una base para el procesamiento de AIoT (Internet de las cosas con inteligencia artificial) y el procesamiento de imágenes en dispositivos de bordes. Incluye una variedad de bibliotecas de funciones a las que los desarrolladores pueden llamar directamente y utiliza la sintaxis de scripts de MicroPython, permitiendo a los desarrolladores editar scripts en tiempo real en su computadora y cargarlos en la placa de desarrollo.



Figura 6.16: IDE microPython.

6.10.1. Instalación de Firmware MaixPy

La instalación es esencial para que los dispositivos funcionen correctamente y comunicarse con elementos de un sistema más amplio, sin interactuar directamente con el usuario.

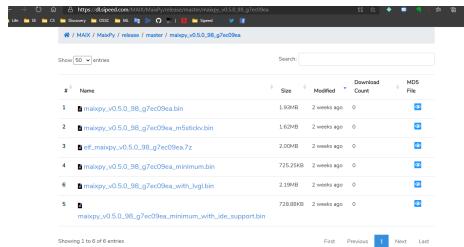


Figura 6.17: Página de instalación de Firmware.

6.10.2. Configuración de kflash-gui

Es una interfaz gráfica de usuario (GUI) para la herramienta kflash.py, que se utiliza para descargar (o quemar) firmware en dispositivos K210. Es una herramienta cruzada de plataforma que permite a los usuarios seleccionar archivos binarios o kfpkg, y establecer la dirección de destino para el firmware. También soporta la fusión de múltiples archivos binarios en uno solo y la conversión de archivos kfpkg a binarios.

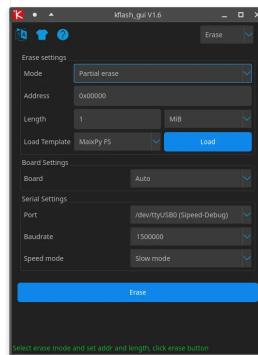


Figura 6.18: Interfaz gráfica de usuario (GUI).

6.11. Interface Maixpy

Se crea el primer código de prueba para observar la conexión de SIPEED MaixDuino y observar el funcionamiento de la cámara en el ide maixpy, donde se estara migrando los algoritmos realizados para la detección de rostros.

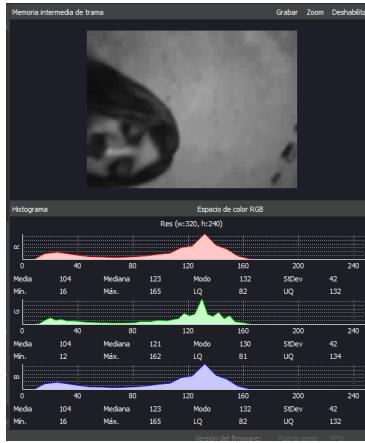


Figura 6.19: Interfaz gráfica de usuario (GUI).

6.12. Creación de robot lina

Se crearon Bocetos de Arquitecturas en las que se pudiera trabajar y que cumplieran las necesidades requeridas. En este caso, se muestran piezas claves que integran su estructura.

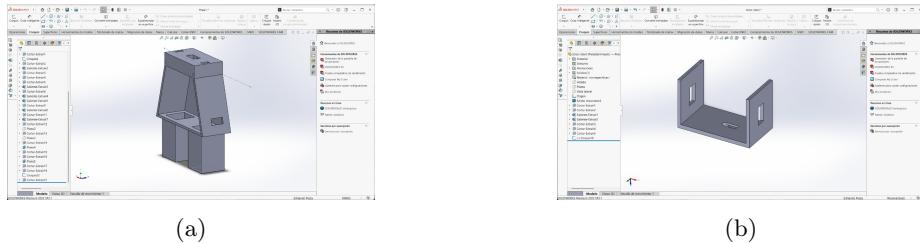


Figura 6.20: Piezas del cuerpo del robot Lina.

El tiempo de impresión se vio afectado debido a diferentes factores como lo puede ser el ambiente o por algún defecto, además cada una de estas piezas cuenta con un relleno diferente, dependiendo de la funcionalidad que emplearán cada una de estas ya que este mismo determinará la dureza de las piezas.

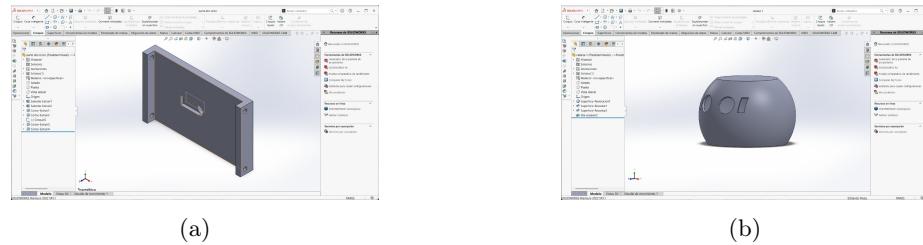


Figura 6.21: Pieza parte superior y cabeza del robot Lina.

El material con el que se estuvo trabajando fue el filamento PLA ya que además de ser resistente se presta para la movilidad que tendrá el robot y permite tener una alta velocidad de impresión a comparación de otros filamentos.

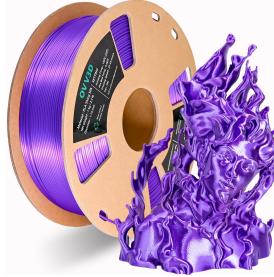


Figura 6.22: Filamento PLA.

Capítulo 7

Resultados

Como primer resultado, se mejoró el algoritmo detector de rostros que se realizó en el cuatrimestre anterior migrandolo a microPython y haciendo ajustes necesarios en las librerías.



Figura 7.1: Detección de rostro en MAixduino.

En esta ocasión se utilizó una esp Maixduino con cámara y lcd para un mejor enfoque y apreciación de el funcionamiento de los algoritmos realizados e implementados como resultados finales.

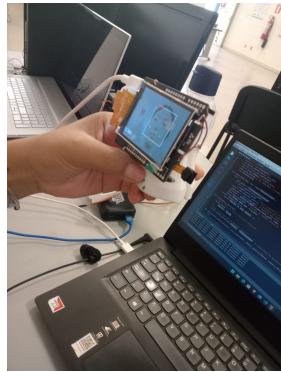


Figura 7.2: Algoritmo funcionando.

Los resultados finales fueron exitosos, logrando que el robot replicara los movimientos con un ligero retraso temporal debido a la estabilidad de la conexión Wi-Fi, la transmisión y procesamiento de los datos.

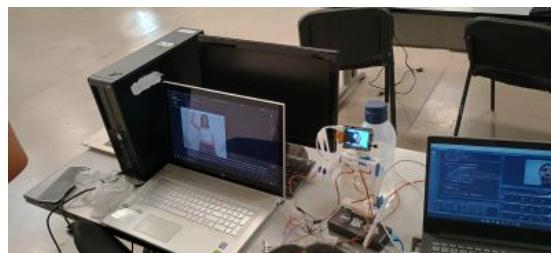


Figura 7.3: Replicación de movimientos exitosa.



Figura 7.4: Movimiento del robot eficiente.

Además, el código desarrollado está optimizado para identificar puntos clave en las articulaciones superiores, incluyendo cadera, hombros, codos y muñecas. Este enfoque permite una sincronización casi precisa de los movimientos del torso y las extremidades superiores, alineándose con los objetivos del proyecto.

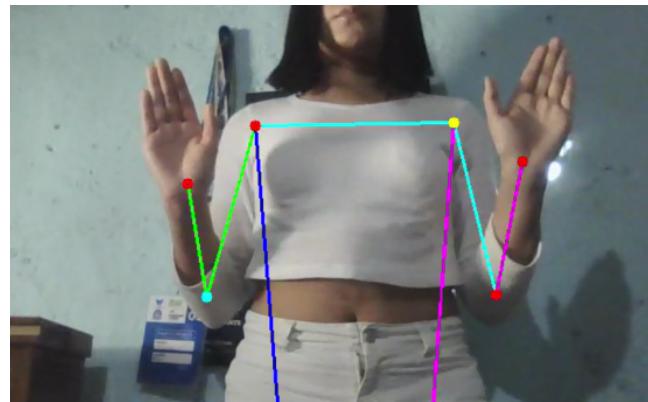


Figura 7.5: Detección de puntos clave con MediaPipe.

En este caso, la importancia de muestra de ángulos es indispensable para observar los parámetros a los que deben de estar sometidos los servomotores ya que esto permite obtener la distancia entre cada punto seguro así mismo permitiendo el cálculo entre cada servomotor para un mejor manejo eficiente dentro de el replicamiento de movimientos.

```
Neck Angle: 252
Left Shoulder Angle: 8
Right Shoulder Angle: 0
Left Elbow Angle: 8
Right Elbow Angle: 0
```

Figura 7.6: Resultados ángulos de distancia.

En la figura 7.7 se puede observar el modelo de impresión terminado donde se aprecia el uso de varios servomotores ya colocados visiblemente en su estructura, permitiendo movimientos articulados y controlados.

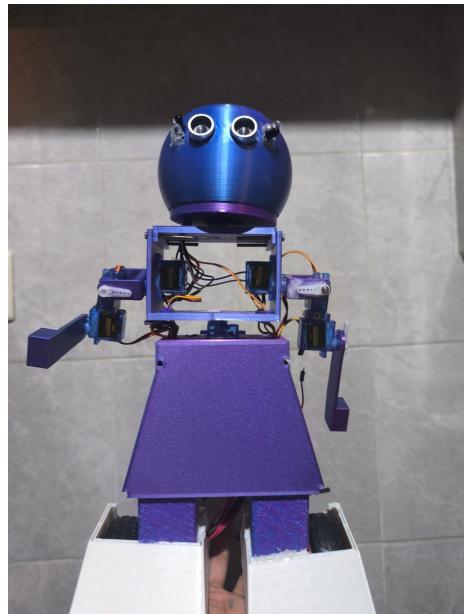


Figura 7.7: Resultado final Robot Lina.

Capítulo 8

Conclusión

En conclusión, la contribución de este proyecto no solo radica en el desarrollo de un esqueleto humanoide robótico, sino también en su potencial para influir en múltiples áreas, desde la investigación y la educación hasta diversas aplicaciones prácticas en la vida cotidiana. Al abordar los desafíos actuales en la robótica humanoide, este proyecto puede facilitar el camino para innovaciones futuras que mejoren la interacción entre humanos y robots. Además, la integración de tecnologías como la comunicación inalámbrica, visión por computadora y el procesamiento en tiempo real refuerza la versatilidad del sistema.

Bibliografía

- [1] *Saenz, J. A. A. (2024, 22 julio). Visión por Computadora (Computer Vision). Inteligencia Artificial Ppt.* <https://inteligenciaartificialppt.com/vision-por-computadora-computer-vision/>
- [2] *Varun, J. (2022, 11 julio). Computer vision using mediapipe and openCV - joel varun - Medium.* Medium. <https://medium.com/@joeajiteshvarun/computer-vision-using-mediapipe-and-opencv-4b44b902b918>
- [3] *ELEGOO US. (s. f.). ELEGOO PLA 3D Printer Filament.* <https://us.elegoo.com/pages/elegoo-pla-3d-printer-filament>