

Práctica 2. Sistema de teleoperación por gestos

Melanie Albaladejo Tasso

24 de mayo de 2024

1. Introducción

La teleoperación robótica representa un campo de estudio en constante evolución que se ha expandido significativamente gracias a los avances recientes en visión por computadora y tecnologías de aprendizaje profundo.

El objetivo de esta práctica es la implementación de un sistema que utiliza el reconocimiento de gestos para controlar un robot de manera intuitiva y precisa, eliminando la necesidad de controles físicos tradicionales que suelen ser menos flexibles y requerir una curva de aprendizaje más pronunciada para los operadores. Utilizando MediaPipe, una solución de código abierto de Google, el sistema es capaz de detectar y procesar gestos en tiempo real, ofreciendo una respuesta inmediata que mejora significativamente la usabilidad y la efectividad de la teleoperación robótica.

2. Estado del arte

La teleoperación robótica y el reconocimiento de gestos son campos que han experimentado un gran desarrollo en la última década, impulsados por avances en tecnologías de visión por computadora y aprendizaje profundo. A continuación, se estudian las principales tecnologías y enfoques utilizados en la teleoperación robótica, especialmente aquellos relacionados con el reconocimiento de gestos humanos.

2.1. Teleoperación Robótica

La teleoperación robótica, que implica el control remoto de robots por operadores humanos, ha sido un campo de investigación y desarrollo en áreas donde la intervención humana directa es impráctica o peligrosa.

Históricamente, estos sistemas dependían de interfaces de control físico, tales como joysticks o paneles gráficos. Aunque funcionales, estos métodos a menudo no son intuitivos y pueden exigir un extenso período de capacitación para los operadores. Sin embargo, con la adopción de la robótica avanzada y los sistemas de control digital en las últimas décadas, la teleoperación ha experimentado notables mejoras en términos de precisión, capacidad de respuesta y funcionalidad general. Además, la integración de tecnologías de realidad virtual y aumentada ha transformado la interfaz de usuario, proporcionando a los operadores una percepción inmersiva y detallada del entorno operativo del robot. Esta evolución ha resultado en un control más preciso y eficaz durante la ejecución de tareas complejas, facilitando operaciones que antes eran consideradas altamente desafiantes.

2.2. Reconocimiento de gestos

El reconocimiento de gestos es una rama de la visión por computadora que se centra en interpretar los movimientos humanos a través de sistemas computacionales. Esta tecnología ha avanzado en los últimos tiempos, gracias al progresos en algoritmos de aprendizaje automático y aumento en la capacidad de procesamiento de los computadores.

El proceso de reconocimiento de gestos comienza con la captura de movimientos mediante cámaras y sensores, seguido por el análisis de estos datos a través de software especializado que interpreta acciones específicas. Este proceso se desarrolla en varias etapas, desde la detección y seguimiento de las manos hasta la clasificación de los gestos realizados. Un ejemplo destacado de herramienta en este campo es MediaPipe, un marco de trabajo desarrollado por Google, que ofrece modelos de detección de gestos altamente eficientes y de código abierto, facilitando así el reconocimiento de gestos complejos [1].

2.3. Tecnologías empleadas en el reconocimiento de gestos para la teleoperación robótica

Entre las tecnologías usadas para la implementación de sistema de teleoperación robótica basados en el reconocimiento de gestos son dos: el aprendizaje profundo y MediaPipe [1]

2.3.1. Aprendizaje profundo en el reconocimiento de gestos

El aprendizaje profundo ha transformado el campo del reconocimiento de gestos, proporcionando herramientas que mejoran la precisión y la eficiencia de estos sistemas. Las redes neuronales profundas, especialmente diseñadas para analizar grandes volúmenes de datos y aprender de ellos, son fundamentales en la interpretación avanzada de los gestos humanos a través de modelos computacionales. Los tipos de redes neuronales más empleados son: redes neuronales convolucionales, redes neuronales recurrentes y LSTM.

Por una parte, las redes neuronales convolucionales analizan imágenes en pequeñas porciones, identificando patrones y características como bordes y texturas, para detectar y clasificar distintos tipos de gestos. Esta capacidad permite extraer características de bajo y alto nivel sin la necesidad de intervención humana directa, mejorando la precisión y la robustez de la detección de gestos [2].

Por otra parte, las redes neuronales recurrentes son empleadas para manejar datos secuenciales, como los gestos que ocurren a lo largo del tiempo. Mientras que las LSTM, una variante avanzada de las RNNs, son particularmente efectivas en recordar información a largo plazo, lo que permite entender secuencias de gestos y predecir acciones futuras basadas en movimientos pasados. Lo que las hace especialmente útil en aplicaciones donde los gestos se realizan en una secuencia que dicta una operación o comando específico [3].

2.3.2. Mediapipe

MediaPipe es un framework de código abierto, desarrollado por Google, diseñado específicamente para facilitar la creación de aplicaciones avanzadas de visión por computadora y soluciones de procesamiento de medios. Este marco incluye una serie de modelos preentrenados que están optimizados para diversas tareas de visión por computadora, tales como la detección de rostros, el seguimiento de manos y el análisis de poses. Una de las ventajas más significativas de MediaPipe es su capacidad para procesar datos en tiempo real, manteniendo un alto nivel de precisión. Esto es importante para aplicaciones que requieren respuestas inmediatas, como los sistemas interactivos y las aplicaciones de realidad aumentada, donde la latencia reducida es esencial para mantener una experiencia de usuario fluida y efectiva.

De entre todos los modelos preentrenados que tiene implementado, el que se va a usar es su modelo de detección de puntos de referencia de la mano. Concretamente, este modelo identifica 21 puntos de referencia distintos en la mano (Figura 1), cada uno correspondiendo a un aspecto clave de la anatomía de la mano, incluyendo las yemas de los dedos, las articulaciones y la palma. La detección de estos puntos no solo captura la posición espacial tridimensional de la mano sino también ofrece información sobre la orientación y la articulación de cada dedo, permitiendo un análisis profundo y detallado de los gestos.

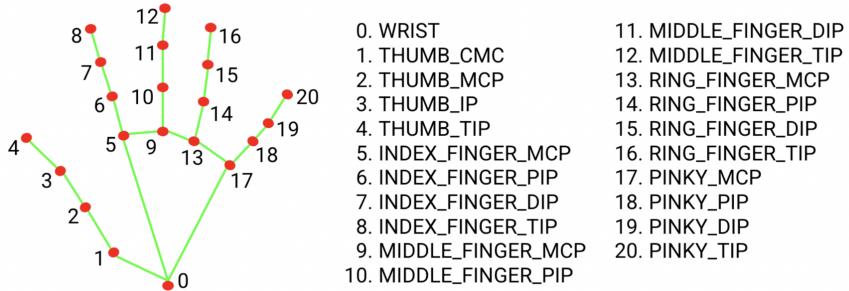


Figura 1: Presentación de los 21 puntos de referencia de la mano

La implementación de este modelo en MediaPipe utiliza una combinación de tecnologías de aprendizaje automático, incluyendo redes neuronales convolucionales y técnicas de visión estéreo, para realizar estimaciones precisas en tiempo real. Estos 21 puntos de referencia son detectados mediante el uso de modelos de aprendizaje profundo que han sido entrenados en un vasto conjunto de imágenes de manos

en diversas posturas y orientaciones.

La detección precisa de estos puntos es de suma importancia para aplicaciones que requieren una interpretación detallada de los gestos, como en sistemas de control por gestos para interfaces de usuario, juegos interactivos, y aplicaciones en rehabilitación médica donde los movimientos precisos de la mano son cruciales. Además, la capacidad de procesar estos datos en tiempo real, con alta fidelidad y bajo retardo, es esencial para aplicaciones interactivas y de realidad aumentada, donde la inmediatez de la respuesta es importante para la experiencia del usuario.

2.4. Aplicaciones en robótica

El reconocimiento de gestos ha revolucionado numerosas aplicaciones robóticas, expandiendo la interacción entre humanos y robots. Algunas de estas áreas son:

- **Navegación de Drones.** La habilidad para controlar drones mediante gestos simplifica su operación, permitiendo a los usuarios dirigir el vuelo con movimientos intuitivos de las manos. Esto es particularmente útil en situaciones donde el uso de controles tradicionales no es viable o donde la rapidez de respuesta es crucial, como en operaciones de búsqueda y rescate o en la cinematografía.
- **Manipulación de objetos por robots industriales.** En entornos industriales, los gestos pueden ser utilizados para controlar robots que realizan tareas de precisión, como el ensamblaje de componentes delicados o la manipulación de materiales peligrosos. Esto no solo mejora la seguridad al mantener a los trabajadores fuera de zonas de riesgo, sino que también aumenta la eficiencia al permitir una interacción más directa y flexible con la maquinaria.
- **Robots de servicio en entornos humanos.** En el sector servicios, los robots pueden utilizar el reconocimiento de gestos para interactuar de manera más natural con las personas, ya sea en tareas de asistencia en hogares, hospitales o en el ámbito de la hospitalidad. Estos robots son capaces de interpretar señales no verbales, lo que facilita una comunicación más efectiva y empática.

2.5. Desafíos actuales

A pesar de los progresos en el campo del reconocimiento de gestos, actualmente existen desventajas a las que tiene que hacer frente.

En primer lugar, la eficacia de la teleoperación depende de la latencia en el procesamiento y de la precisión en la detección de gestos. Retrasos mínimos en la respuesta del sistema o errores en la interpretación de gestos pueden tener consecuencias graves, especialmente en aplicaciones como la cirugía robótica o el control de vehículos autónomos [4].

En segundo lugar, los sistemas de reconocimiento de gestos deben ser robustos a variaciones en el entorno, como cambios en la iluminación o en las condiciones meteorológicas. La eficiencia de estos sistemas puede verse comprometida en ambientes con iluminación pobre o con elementos que obstruyan la visibilidad, lo cual es un desafío constante para los ingenieros [4].

Por último, el diseño de las interfaces deben ser lo suficientemente intuitivas para que usuarios sin experiencia puedan operarla con poco o ningún entrenamiento, pero también debe ser capaz de realizar tareas complejas sin comprometer su funcionalidad [5].

3. Desarrollo

En este apartado se procede a explicar como se han ido modificando los archivos para implementar el sistema de teleoperación por gestos. Hay que tener a bien que los archivos modificados han sido cuatro: show_camera_robot.py, capture_video.py, pose_estimation.py y obstacle_avoidance.py.

3.1. Visualización de la cámara del entorno

La visualización de la cámara del entorno se maneja a través de ROS, donde el archivo `show_camera_robot.py` establece un nodo que se suscribe al tópico correspondiente a la cámara del entorno. Este nodo utiliza bibliotecas como OpenCV para capturar y procesar las imágenes de vídeo. El tópico específico al que se suscribe es `/camera/color/image_raw`, que proporciona imágenes sin procesar directamente desde la cámara. Para realizar la suscripción se declara en el código `rospy.Subscriber('/camera/color/image_raw', Image, image_callback)`.

La visualización de la cámara del entorno en el sistema de teleoperación se gestiona a través de ROS. En este proceso, el archivo `show_camera_robot.py` configura un nodo que se suscribe al tópico `/camera/color/image_raw`. Este tópico transmite imágenes sin procesar directamente desde la cámara, proporcionando datos visuales crudos que son esenciales para la monitorización en tiempo real. Para la captura y procesamiento de estas imágenes, se emplea la biblioteca OpenCV, y la suscripción al tópico se realiza mediante la declaración en el código: `rospy.Subscriber('/camera/color/image_raw', Image, image_callback)`.

Esto hace que el sistema proporcione una interfaz gráfica de usuario (GUI) donde las imágenes procesadas se muestran en tiempo real. Permitiendo así al operador tener una visión directa y continua del entorno del robot, facilitando la toma de decisiones y la dirección del robot con precisión y eficiencia.

3.2. Obtención de la imagen en el operador

En esta sección del desarrollo, se implementa la captura de la imagen del operador. Para ello, se establece un nodo en ROS que maneja la captura de vídeo desde la webcam. Este nodo es el responsable de publicar las imágenes capturadas en un tópico específico, `/operator/image`, que luego será utilizado por otros módulos del sistema para el análisis de gestos.

Inicialmente, se establece un publicador en el tópico `/operator/image` mediante la declaración en el código: `pub_image = rospy.Publisher('/operator/image', Image, queue_size=10)`. Esta configuración permite que las imágenes capturadas sean accesibles para otros nodos dentro de la red ROS, facilitando la integración y el análisis posterior de los datos.

A continuación, se define la captura de vídeo a través de la biblioteca OpenCV, configurando el dispositivo de captura de la siguiente manera: `cap = cv2.VideoCapture(0)`. Este comando inicializa la webcam conectada al sistema y se verifica si está operativa mediante la comprobación de su estado, asegurando que el dispositivo esté disponible para la captura de video.

```
while not rospy.is_shutdown():
    # Captura un frame de la webcam
    ret, frame = cap.read()
    if not ret:
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            rospy.logerr("No se pudo abrir la cámara.")
            return
        continue

    # Convierte el frame de OpenCV a un mensaje ROS
    ros_image = bridge.cv2_to_imgmsg(frame, "bgr8")

    # Publica el mensaje en el tópico
    pub_image.publish(ros_image)

    # Espera para cumplir con la tasa de publicación
    rate.sleep()

    # Cuando termines, libera la captura
    cap.release()
```

Figura 2: Lógica de transmisión de información de la webcam a ROS

Tras esto, como se muestra en la Figura 2, el sistema procede a capturar continuamente cada frame

de vídeo. Cada frame capturado se convierte a un mensaje de ROS utilizando el puente entre OpenCV y ROS: `bridge.cv2_to_imgmsg(frame, "bgr8")` y este mensaje es luego publicado en el tópico /operator/image definido previamente.

3.3. Captura e interpretación de los gestos

En este apartado se implementa la lógica de captura y análisis de gestos del operador. Este archivo establece un nodo en ROS que procesa las imágenes obtenidas del operador para detectar y interpretar gestos en tiempo real. El cual se inicializa mediante: `rospy.init_node('pose_estimation', anonymous=True)`.

Inicialmente, se configura el sistema de detección de gestos de mano utilizando la biblioteca MediaPipe. Específicamente, se importan dos módulos clave: `mp.solutions.hands` y `mp.solutions.drawing_utils`. El primero, `mp.solutions.hands`, es un módulo especializado de MediaPipe que se encarga de la detección de manos. Este módulo incluye todas las funciones necesarias para inicializar y desplegar el modelo de detección de manos, así como para procesar imágenes y detectar manos en ellas de manera eficiente. Por otro lado, `mp.solutions.drawing_utils` se utiliza para la visualización de los resultados del proceso de detección. Este módulo proporciona herramientas útiles para dibujar los puntos de referencia detectados en las manos sobre las imágenes, facilitando la visualización y el debug del sistema.

A continuación, el detector de manos se inicializa con configuraciones específicas para optimizar su funcionamiento en escenarios dinámicos. La configuración `mp_hands.Hands(static_image_mode=False, max_num_hands=1, min_detection_confidence=0.5, min_tracking_confidence=0.5)` establece el detector para funcionar en modo de vídeo, lo que implica que el sistema espera un flujo continuo de imágenes y utiliza internamente modelos optimizados para el seguimiento en tiempo real. Además, se configura para detectar y seguir únicamente una mano. Los umbrales de confianza para la detección y el seguimiento se establecen en 0.5, lo que significa que el sistema requiere una probabilidad de al menos el 50% para validar la detección y el seguimiento de la mano, asegurando un equilibrio entre precisión y rendimiento.

```
#Procesar la imagen del operador
def image_callback(msg):
    bridge = CvBridge()
    try:
        # Convertir la imagen de ROS a una imagen de OpenCV
        cv_image = bridge.imgmsg_to_cv2(msg, "bgr8")
    except CvBridgeError as e:
        print(e)

    #Procesar la imagen con MediaPipe
    cv_image_processed = cv2.cvtColor(cv_image, cv2.COLOR_BGR2RGB)
    results = hands.process(cv_image_processed)

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            # Dibujar los landmarks sobre la imagen
            mp_drawing.draw_landmarks(
                cv_image, hand_landmarks, mp_hands.HAND_CONNECTIONS)

            # Reconocer el gesto mediante alguna clasificación a partir de los landmarks
            gesture = gesture_classify(hand_landmarks)
            rospy.loginfo("Detected gesture: %s" % gesture)

            # Interpretar el gesto obtenido y enviar la orden de control ackermann
            ackermann_msg = ackermann_msgs.msg.AckermannDrive()

            if gesture == 1:
                ackermann_msg.speed = 2.0 # Avanzar
                ackermann_msg.steering_angle = -1.0 # Girar a la derecha
            elif gesture == 2:
                ackermann_msg.speed = 2.0 # Mover Recto
                ackermann_msg.steering_angle = 1.0 # Girar a la izquierda
            elif gesture == 3:
                ackermann_msg.speed = 2.0 # Avanzar
                ackermann_msg.steering_angle = 0.0 # Recto
            else:
                ackermann_msg.speed = 0.0 # Detener
                ackermann_msg.steering_angle = 0.0 # Recto

            ackermann_command_publisher.publish(ackermann_msg)

    # Mostrar la imagen con los landmarks/gestos detectados
    cv2.imshow("Hand pose Estimation", cv_image)
    cv2.waitKey(1)
```

Figura 3: Implementación función `image_callback` del fichero `pose_estimation.py`

El reconocimiento de gestos y la definición de los movimientos que debe hacer el robot ackermann

para cada uno de ellos comienza cuando el suscriptor del tópico recibe un mensaje ROS. La función `image_callback`, cuya implementación se muestra en la Figura 3, es la encargada de procesar las imágenes del operador, reconocer los gestos de las manos utilizando la biblioteca MediaPipe, y enviar los comandos basados en los gestos reconocidos para controlar un robot mediante un sistema de conducción Ackermann.

En primer lugar, se convierte la imagen del formato ROS al formato 'bgr8' de OpenCV para que puedan ser procesada empleando esta librería. Esta transformación se realiza mediante `bridge.imgmsg_to_cv2(msg, "bgr8")`. Además, para que la imagen pueda ser procesada por el modelo de detección de manos de MediaPipe configurado previamente mediante `hands.process`, la imagen debe convertirse al formato requerido por MediaPipe que es RGB.

A continuación, se compueba si los landmarks de la mano fueron detectados. Si es así, cada conjunto de landmarks detectados se dibuja sobre la imagen original para su visualización y basándonos en estos landmarks, se procede a realizar la clasificación de los gestos con la llamada a la función `gesture_classify`.

La función `gesture_classify` está diseñada para clasificar los gestos definidos en el Cuadro 1, donde se especifican los gestos seleccionados para acciones como *girar a la derecha*, *girar a la izquierda*, y *moverse hacia delante*.



Cuadro 1: Gestos de movimientos definidos

La lógica para la detección de estos gestos se presenta en la Figura 4, que muestra cómo se analiza la posición en el eje X de las partes TIP y DIP, o la IP para el pulgar, de cada dedo. En particular, se considera que un dedo está extendido si la posición en el eje X de la TIP es inferior a la de su correspondiente DIP. Basándose en esto, la función determina:

- *Girar a la derecha*. Este gesto se detecta si solo el pulgar y el dedo índice están extendidos.
- *Girar a la izquierda*. Se requiere que el pulgar, el dedo índice y el dedo corazón estén extendidos.
- *Moverse hacia delante*. Se identifica cuando todos los dedos de la mano están extendidos.

Si los dedos presentan cualquier otra combinación de posiciones, no se detectará ninguno de estos gestos predefinidos, y como resultado, no se emitirá ningún comando de movimiento. Esta configuración asegura que sólo los gestos específicamente definidos y reconocidos desencadenen una respuesta en el sistema de control del robot.

```
def gesture_classify(hand_landmarks):
    thumb_opened = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x < hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_DIP].x
    index_opened = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].x < hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_DIP].x
    middle_opened = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].x < hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_DIP].x
    ring_opened = hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TIP].x < hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DIP].x
    pinky_opened = hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].x < hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].x

    if index_opened and not middle_opened and not ring_opened and not pinky_opened and thumb_opened:
        return 1 #"Girar Derecha"
    elif index_opened and middle_opened and not ring_opened and not pinky_opened and thumb_opened:
        return 2 #"Girar Izquierda"
    elif thumb_opened and index_opened and middle_opened and ring_opened and pinky_opened:
        return 3 #"Recto"
    else:
        return 0 #"Parar"
```

Figura 4: Implementación de la función `gesture_classify`

Una vez que el gesto ha sido reconocido, se configura un mensaje de tipo AckermannDrive con los parámetros de velocidad y ángulo de dirección correspondientes. Concretamente, el valor de los parámetros seleccionados para cada movimiento son los que se muestran en el Cuadro 2.

Gesto	Velocidad	Ángulo de dirección
Girar a la derecha	2.0	-1.0
Girar a la izquierda	2.0	1.0
Moverse hacia delante	2.0	0.0
Parar	0.0	0.0

Cuadro 2: Valores de velocidad y ángulo de dirección para cada gesto

Por último, este mensaje se publica en un tópico de ROS para que otros nodos puedan recibirla y actuar en consecuencia.

3.4. Sistema de seguridad del robot

Por último, se diseña un nodo en ROS cuyo objetivo es garantizar la seguridad operativa del robot al integrar la toma de decisiones basada en la información de obstáculos detectados. Este sistema evita colisiones y asegura una operación segura en el entorno en el que el robot se desplaza.

El sistema de seguridad del robot recibe órdenes de control del sistema de teleoperación y las procesa junto con la información de detección de obstáculos para tomar decisiones de movimiento seguras. Este nodo de ROS se suscribe al tópico /blue/preorder_ackermann_cmd, donde recibe las órdenes preliminares de control basadas en los gestos detectados.

Antes de ejecutar cualquier comando de movimiento, el sistema consulta otro tópico, /obstacles, que contiene datos sobre posibles obstáculos detectados en el entorno. Basándose en estos datos, el sistema decide si es seguro proceder con el movimiento solicitado o si es necesario modificar la acción para evitar una colisión. La implementación de esto se muestra en la Figura 5.

```
def obstacle_callback(self, msg):
    # Procesar la nube de puntos con los obstáculos teniendo en cuenta el último
    points = list(pc2.read_points(msg, skip_nans=True))

    for point in points:
        x, y, z = point[:3]
        if -1.5 < y < 1.5 and -1.5 < x < 1.5:
            self.obstacle_detected = True

    cmd = self.last_ackermann_cmd
    if self.obstacle_detected:
        rospy.loginfo("Parar el robot - Obstáculo detectado.")
        cmd.speed = 0.0
        cmd.steering_angle = 0.0

    # Modificar mensaje de ackermann si es necesario
    cmd = self.last_ackermann_cmd
    self.ackermann_command_publisher.publish(cmd)

    return
```

Figura 5: Implementación de lógica de evitación de obstáculos

Si se detecta un obstáculo en la trayectoria planeada del robot, el sistema generará una orden para que el robot se detenga. Mientras que si el camino está libre de obstáculos, el sistema permitirá que se ejecute la orden de control original.

Una vez tomada la decisión basada en la evaluación de seguridad, el sistema publica la orden definitiva de movimiento en el tópico /blue/ackermann_cmd. Este es el tópico que el robot escucha para recibir comandos de movimiento final.

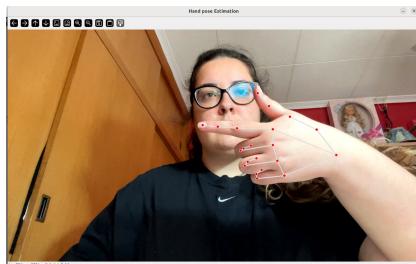
4. Experimentación

Una vez que se ha explicado como se ha implementado todo el código se procede a exponer los resultados conseguidos cuando se ejecuta este. En primer lugar, se muestra en la Figura 6 un ejemplo de la visualización desde la cámara del entorno.

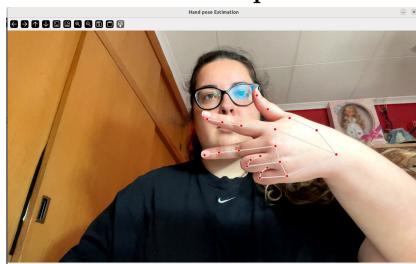


Figura 6: Visualización cámara del entorno

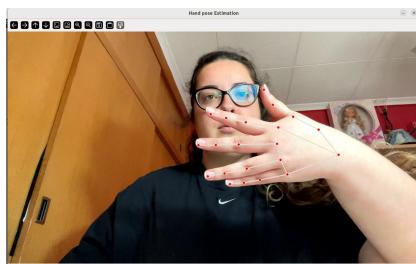
Girar a la derecha



Girar a la izquierda



Moverse hacia delante



Cuadro 3: Detección de los landmarks de la mano para los diferentes gestos definidos

A continuación, en el Cuadro 3 se muestra la detección de los landmarks de la mano para cada

uno de los gestos implementados relacionados con una acción determinada, es decir, *girar a la derecha*, *girar a la izquierda*, y *moverse hacia delante*.

Una vez expuesto esto, se procede a ilustrar los movimientos ejecutados por el robot. Es importante mencionar que el vídeo adjunto a la práctica ofrece una visualización más clara de estos. En particular, en el Cuadro 4, se detalla cómo el robot responde a distintos gestos de comando: al realizar el gesto de girar a la derecha, la orientación del robot se ajusta para desplazarse hacia la derecha; similarmente, con el gesto de girar a la izquierda, el robot se orienta hacia la izquierda; y finalmente, cuando se indica seguir recto, el robot mantiene una trayectoria directa y estable.



Cuadro 4: Ejemplicación de los movimientos correspondientes a cada gesto

Por último, se demuestra la evitación de los obtáculos con la imagen que se muestra en la Figura 7. A pesar de que claramente no se puede observar el movimiento, cuando se está cerca del radio de detección de obstáculos a pesar de que el gesto que se realice sea seguir hacia delante el robot se mantiene parado.

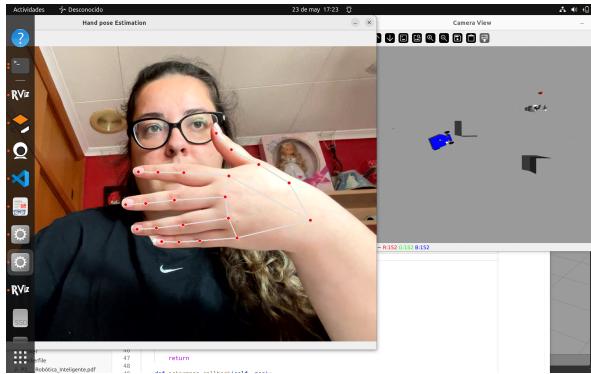


Figura 7: Ejemplo de parada cuando se detecta un obtáculo

5. Conclusiones

El desarrollo del sistema de teleoperación por gestos proporciona una interfaz intuitiva para la manipulación remota de robots en entornos complejos. A través del uso de tecnologías de reconocimiento de gestos y aprendizaje profundo, se ha logrado una interacción natural y directa, lo que ha mejorado la precisión y la seguridad operacional. La integración de un sistema de seguridad también ha asegurado que el robot opere de manera segura dentro de su entorno, adaptándose a obstáculos y condiciones de este.

A pesar del correcto funcionamiento de la práctica, existen problemas, como la latencia en el procesamiento y la necesidad de mejorar la robustez del sistema incorporando nuevos gestos que, por ejemplo, controlen el movimiento del robot.

Enlace a GitHub

A continuación, se adjunta el enlace a GitHub donde está el código implementado, el vídeo demostrativo y también, una copia de la memoria del proyecto.

[GitHub - Melanie Albaladejo Tasso](#)

Referencias

- [1] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking, 2020.
- [2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [3] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.
- [4] Sergio Escalera, Vassilis Athitsos, and Isabelle Guyon. Challenges in multi-modal gesture recognition. *Gesture recognition*, pages 1–60, 2017.
- [5] Hongyu Zhou, Dongying Wang, Yang Yu, and Zhenrong Zhang. Research progress of human-computer interaction technology based on gesture recognition. *Electronics*, 12(13):2805, 2023.