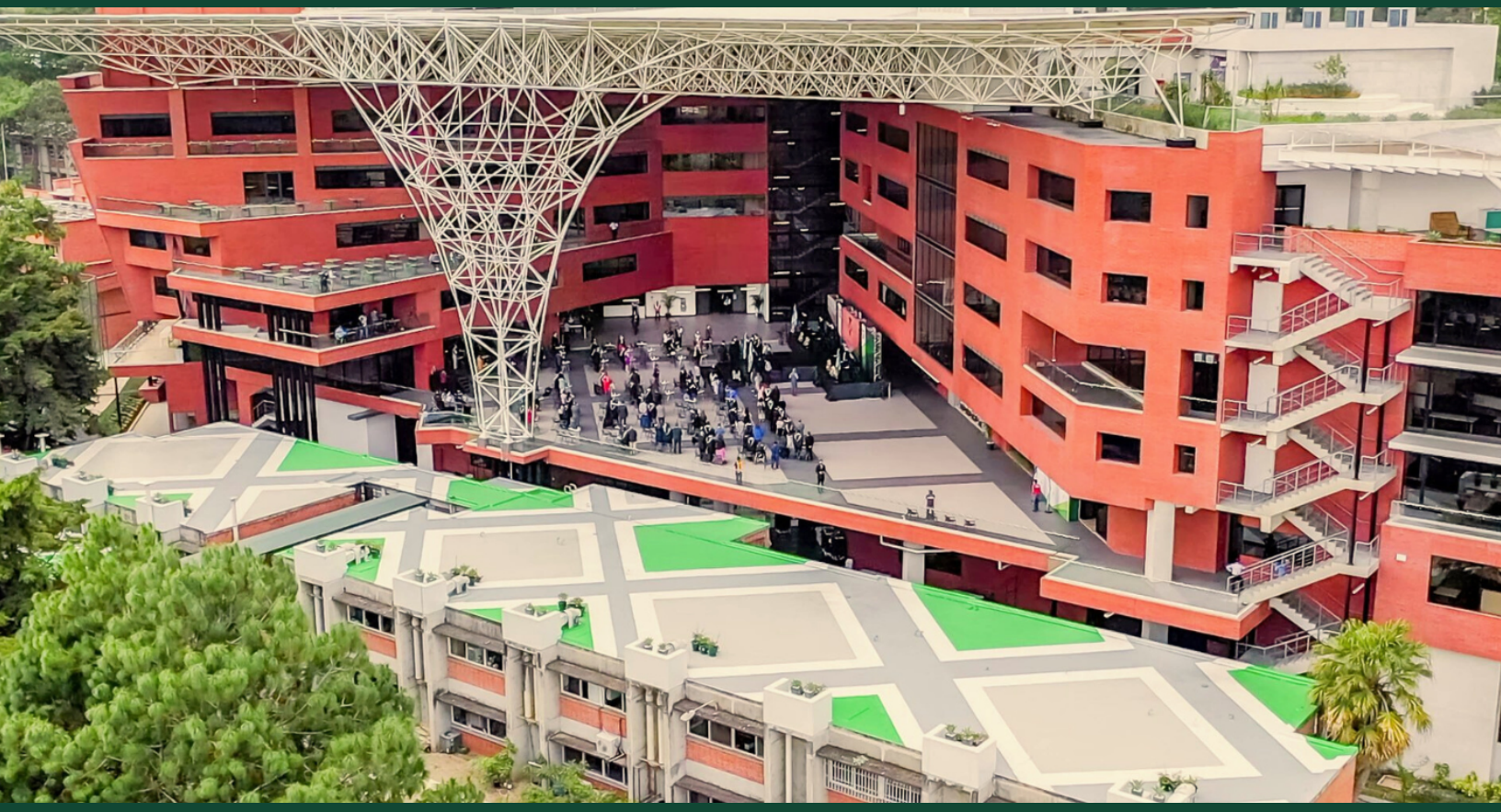
Desarrollo de un sistema para el control de dispositivos robóticos por medio de gestos y un sensor Leap Motion

Ana Marcela Padilla Micheo



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

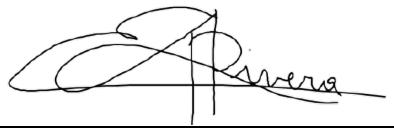
Desarrollo de un sistema para el control de dispositivos robóticos por medio de gestos y un sensor Leap Motion

Trabajo de graduación presentado por Ana Marcela Padilla Micheo para optar al grado académico de Licenciada en Ingeniería Mecatrónica

Guatemala,

2025

Vo.Bo.:

(f) 

Dr. Luis Alberto Rivera Estrada

(f) 

M.Sc. Carlos Alberto Esquit Hernández

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Prefacio

El presente trabajo de graduación es el resultado de un proceso de aprendizaje, investigación y desarrollo que no habría sido posible sin el apoyo incondicional de diversas personas e instituciones.

En primer lugar, agradezco a Dios, quien ha sido la fortaleza de mi corazón y mi mente, cuya guía me permitió alcanzar este sueño.

Agradezco a mi madre, Julieta, por ser siempre mi refugio y respaldo en cada etapa; a mi padre, Marvin, por orientarme con su ejemplo y apoyarme siempre; a mi hermana, Kimberly, por inspirarme a ser una mujer que trasciende lo convencional; y a mi hermano, Marvin, por motivarme a estudiar Ingeniería Mecatrónica. Extiendo también mi gratitud a mis sobrinos, Mathéo y Antoine, por su apoyo emocional, así como a mi novio, cuya compañía y paciencia estuvieron en todo momento. Del mismo modo, agradezco a mis tíos, cuñados, primos y amigos, cuyo respaldo fue siempre constante, en especial a mis cuñados Luisa y Jean-Roch, por su apoyo incondicional. De manera especial, guardo en mi corazón a mi abuelita "José", quien, aunque ya no está físicamente conmigo, dejó enseñanzas y recuerdos que iluminan cada paso de este logro.

Expreso también un profundo agradecimiento a mi asesor, Dr. Luis Rivera, por su orientación, compromiso y valiosas sugerencias técnicas, que fueron esenciales para culminar con éxito este trabajo. Asimismo, reconozco la ayuda y acompañamiento del profesor MSc. Miguel Zea en este proceso.

Finalmente, agradezco a la Universidad del Valle de Guatemala por abrirme sus puertas, formarme como profesional y brindarme la oportunidad de crecer tanto en lo académico como en lo personal.

Índice general

Índice de figuras

Resumen

El presente trabajo de graduación desarrolla un sistema de control gestual basado en el sensor Leap Motion para la operación de distintos dispositivos robóticos, tanto en entornos simulados como físicos. El problema central identificado es la limitada interacción intuitiva en actividades educativas de divulgación científica, particularmente en talleres STEM realizados en la Universidad del Valle de Guatemala. Para atender esta necesidad, se propone una interfaz natural que permita controlar robots mediante movimientos de la mano sin contacto físico.

El proyecto tiene como objetivos principales investigar las características del sensor, extraer y procesar la información gestual en tiempo real, traducirla a comandos para robots disponibles en la UVG y diseñar una interfaz gráfica que facilite su uso. La metodología adoptada fue incremental: Se inició con simulaciones básicas mediante un agente puntual, avanzando luego hacia modelos más complejos como una mano virtual, el robot Pololu 3Pi+ en Webots y el robot Sawyer en Matlab. Finalmente, se integraron sistemas físicos como una mano animatrónica y el Pololu 3Pi+ físico. Para asegurar el funcionamiento del sistema, se emplearon diversos métodos de comunicación entre procesos, incluyendo UDP, comunicación serial y archivos temporales.

Los resultados demuestran la viabilidad del uso de Leap Motion como interfaz de control para múltiples sistemas robóticos, logrando replicar acciones gestuales de forma intuitiva y funcional. Asimismo, se elaboraron talleres y una guía técnica para facilitar la replicación del sistema en actividades educativas, ampliando así su impacto dentro de la comunidad académica.

Palabras clave: control gestual, robótica, Leap Motion, interacción humano-robot, STEM.

Abstract

This graduation project presents the development of a gesture–based control system using the Leap Motion sensor to operate various robotic devices, both in simulated and physical environments. The central problem identified is the limited intuitive interaction available in educational outreach activities, particularly in STEM workshops held at the Universidad del Valle de Guatemala. To address this need, the project proposes a natural interface that enables robot control through hand movements without physical contact.

The main objectives of the project include investigating the characteristics of the sensor, extracting and processing gesture information in real time, translating this information into commands for the robots available at UVG, and designing a graphical interface that facilitates user interaction. The methodology followed an incremental approach: it began with basic simulations using a point agent, then progressed to more complex models such as a virtual hand, the Pololu 3Pi+ robot in Webots, and the Sawyer robot in MATLAB. Finally, physical systems such as the animatronic hand and the Pololu 3Pi+ robot were integrated. To ensure proper system operation, several interprocess communication methods were employed, including UDP, serial communication, and temporary files.

The results demonstrate the feasibility of using the Leap Motion sensor as a control interface for multiple robotic systems, successfully replicating gesture–based actions in an intuitive and functional manner. Additionally, workshops and a technical guide were developed to support the replication of the system in educational activities, thereby expanding its impact within the academic community.

Keywords: gesture control, robotics, leap motion, human–robot interaction, stem.

CAPÍTULO 1

Introducción

Este trabajo se enmarca en el área de la ingeniería mecatrónica, con un enfoque en el estudio e implementación de interfaces para el control de sistemas robóticos. En los últimos años, el uso de sensores como Leap Motion ha ganado importancia al facilitar la detección precisa de gestos y posiciones de la mano, abriendo nuevas posibilidades para la interacción entre humanos y máquinas.

El objetivo principal de este proyecto fue desarrollar un sistema que permitiera controlar diferentes dispositivos robóticos mediante gestos detectados por el sensor Leap Motion, con el fin de aumentar la interactividad en entornos educativos y de divulgación científica, especialmente en actividades STEM en la Universidad del Valle de Guatemala. De esta manera, se busca incentivar el interés por la robótica entre estudiantes y futuros aspirantes a la carrera de Ingeniería Mecatrónica.

El alcance del proyecto abarca el diseño, la implementación y la validación de un conjunto de sistemas robóticos controlados por gestos, que incluyen plataformas simuladas (mano virtual, esfera en Matlab, robot Pololu en Webots y robot Sawyer) y dispositivos físicos (mano animatrónica y robot Pololu 3Pi+). El enfoque no era desarrollar hardware desde cero, sino integrar herramientas a través de software y comunicación entre procesos para construir un sistema funcional y replicable.

La metodología aplicada fue incremental: inicialmente, se investigaron las capacidades de Leap Motion y se realizaron pruebas en simulaciones simples, como una esfera virtual; luego, se avanzó a modelos más complejos, como una mano simulada y el robot Pololu 3Pi+ en Webots; finalmente, se validaron los resultados en sistemas físicos. Para la comunicación entre dispositivos, se utilizaron diversos métodos, como UDP, comunicación serial y archivos temporales, seleccionando la alternativa más adecuada para cada plataforma. Además, se diseñó una interfaz gráfica en Python que centraliza la gestión de los subsistemas, y se elaboraron talleres prácticos con una guía técnica para su uso en actividades académicas.

CAPÍTULO 2

Antecedentes

En los últimos años, la tecnología de los sensores ha logrado avances significativos, permitiendo una interacción más intuitiva entre humanos y máquinas. Estos avances han abierto nuevas posibilidades para el desarrollo de sistemas en el campo biomédico. En este contexto, se han explorado diversas soluciones tecnológicas que emplean sensores como el Leap Motion para interpretar gestos y movimientos de la mano, lo que ha impulsado el desarrollo de distintas interfaces.

2.1. Trabajos desarrollados con Leap Motion

Diversas investigaciones han explorado el uso del sensor Leap Motion para el reconocimiento de gestos de la mano y su aplicación en diferentes sistemas de interacción. A continuación, se presentan tres estudios representativos que demuestran el potencial de esta tecnología en tareas de reconocimiento, control e interpretación de movimientos.

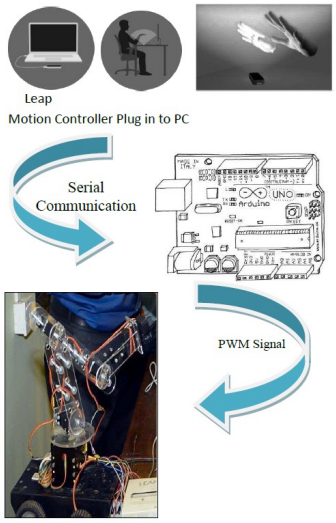
Uno de los trabajos más destacados es el artículo *Hand Gesture Recognition with Leap Motion*, desarrollado por investigadores de la Universidad de Tecnología de Dalian. En este estudio, se propone una combinación de datos de seguimiento y características de imágenes obtenidas por sensores para mejorar la precisión del reconocimiento de gestos. En particular, se introduce una nueva métrica llamada *Fingertips Tip Distance* (T), la cual, al integrarse con descriptores *Histogram of Oriented Gradients* (HOG) extraídos de las imágenes del sensor, permite mejorar el desempeño general del sistema. Como resultado, se logró una mejora notable en la precisión del reconocimiento de gestos, aunque en el artículo señalan que es necesario continuar con investigaciones futuras para seguir optimizando su desempeño [1].

El artículo *Gesture Recognition with the Motion Controller*, presentado por el *Rochester Institute of Technology* (RIT), trabaja en el uso del sensor Leap Motion para implementar un sistema de reconocimiento de gestos de la mano basado en redes neuronales. Utilizaron la

librería de Matlab llamada *MatConvNet* para implementar una *Convolutional Neural Network* (CNN) capaz de identificar cinco gestos distintos. La recolección de datos se realizó a través de una interfaz gráfica, la cual grababa las señales cuando los participantes presionaban la tecla 's' mientras ejecutaban los gestos. Una limitación importante de este estudio fue que únicamente se consideraron gestos que variaban en el plano XY, lo cual restringe el análisis a movimientos bidimensionales [2].

Otro caso es el artículo *Design of Control System for Articulated Robot Using Leap Motion Sensor*, en el cual se presenta un sistema para controlar un brazo robótico articulado mediante gestos capturados con Leap Motion. El sensor detecta los movimientos de la mano, los cuales son procesados en una computadora y posteriormente enviados, a través de una interfaz programada en *Java*, a un microcontrolador Arduino Uno (Figura 1). Este dispositivo ajusta los servomotores del brazo robótico para imitar los movimientos detectados. Los resultados obtenidos muestran una buena precisión y bajo retardo en la respuesta del sistema. Sin embargo, se identificaron limitaciones como una frecuencia de muestreo inestable que podría afectar el control en tiempo real y un rango de operación limitado (hasta 1 metro) [3].

Figura 1. Sistema propuesto por IRJET.



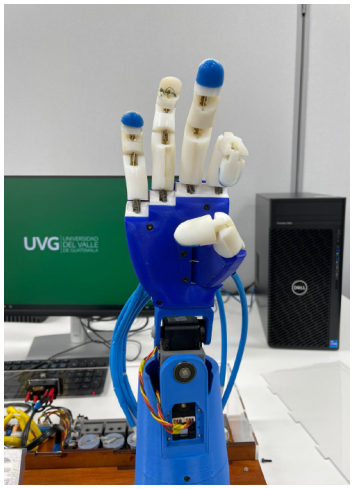
Nota. La imagen muestra el sistema propueto para controlar un brazo robótico. Esta imagen fue obtenida de [3].

2.2. Trabajos desarrollados en la UVG

En la Universidad del Valle de Guatemala (UVG), se han desarrollado diversos proyectos relacionados con gestos para el control de sistemas robóticos, enfocados principalmente en interfaces biomédicas. La primer interración con este tipo de sistemas fue presentado por María Fernanda Girón en 2020 [4], quien presentó una interfaz para el control de sistemas robóticos mediante señales electromiográficas de superficie (sEMG).

Por otro lado, Santiago Jose Rivera Lemus [5] desarrolló en 2025 una interfaz biomédica y sensores inerciales para el control de actuadores en tiempo real. Su trabajo, que utiliza Matlab como herramienta principal, se basa en gestos predeterminados y emplea la mano animatrónica (Figura 2). Entre las limitaciones de este proyecto destaca la complejidad de su integración, mientras que sus fortalezas incluyen el uso de dos sensores (BITalino y Myowave) los cuales ofrecen una mayor precisión.

Figura 2. Mano animatrónica usada en el trabajo de Santiago Rivera.



Nota. La imagen muestra la mano animatrónica con el conteo del número 3 en el proyecto de Santiago Rivera. Esta imagen fue obtenida de [5].

Adicionalmente, el proyecto de graduación de Saby Andrade [6], que constituye uno de los antecedentes directos de este trabajo, desarrolla una interfaz biomédica para controlar agentes robóticos móviles mediante señales EMG, EEG y EOG. El estudio integra adquisición, filtrado, extracción de características y clasificación con SVM, validando el control en tiempo real tanto en simuladores como en robots físicos dentro de la plataforma Robotat.

CAPÍTULO 3

Justificación

Actualmente en la Universidad del Valle de Guatemala (UVG) se llevan a cabo diversos eventos para atraer estudiantes interesados en formar parte de la institución. Entre estos eventos destacan la Experiencia UVG, Curso de Vacaciones Mujeres en Ingeniería, Open House, múltiples talleres de robótica entre otras actividades *Science, Technology, Engineering and Mathematics* (STEM). Por lo tanto, este proyecto tiene como objetivo incrementar la interactividad y dinamismo en los talleres impartidos durante estos eventos mediante el control de sistemas robóticos utilizando un sensor Leap Motion. Esta iniciativa no sólo facilitará la participación de distintos estudiantes, sino que también ofrecerá una experiencia educativa más atractiva.

La utilización de un sensor Leap Motion es esencial para este proyecto porque posee una alta precisión, lo que permitirá que futuros proyectos tengan un alcance mayor y sean llamativos. El uso de este sensor en los talleres permitirá despertar la curiosidad científica brindando actividades innovadoras y efectivas. Su uso en la Experiencia UVG y otros eventos similares potenciará el impacto de lo talleres y contribuiría a mayor interés por la robótica.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar un sistema que permita controlar dispositivos robóticos por medio de gestos, utilizando un sensor Leap Motion.

4.2. Objetivos específicos

- Investigar las características y funcionalidad del sensor Leap Motion.
- Extraer y analizar los datos de gestos de la mano proporcionados por el sensor Leap Motion.
- Traducir la información proporcionada por el sensor Leap Motion a comandos para el control de dispositivos robóticos disponibles en la UVG.
- Diseñar e implementar una interfaz gráfica que le facilite a los usuarios el uso del sistema de control desarrollado.
- Diseñar un taller que pueda realizarse en eventos de la UVG basado en el sistema desarrollado.

CAPÍTULO 5

Alcance

El presente trabajo de graduación se limita al diseño, implementación y validación de un sistema para el control de dispositivos robóticos mediante gestos de la mano, utilizando el sensor Leap Motion. El desarrollo se centró en integrar este dispositivo con diferentes plataformas de software y hardware disponibles en la Universidad del Valle de Guatemala, sin modificar la estructura interna del sensor ni desarrollar hardware propio desde cero.

El proyecto abarca tanto entornos simulados como físicos. En el ámbito de la simulación, se implementaron una mano virtual en Matlab, una esfera controlada por gestos, el robot móvil Pololu 3Pi+ en Webots y el robot Sawyer en Matlab. En la parte física, se integró el control con una mano animatrónica previamente desarrollada en la UVG y con el robot Pololu 3Pi+ real. En todos los casos, la comunicación entre plataformas se realizó mediante Python como lenguaje principal, utilizando protocolos como UDP, comunicación serial y archivos temporales, según la necesidad de cada integración.

El alcance del trabajo incluye también el desarrollo de una interfaz gráfica en Python que centraliza la ejecución de los diferentes sistemas robóticos, así como la elaboración de talleres y una guía técnica que facilitan la replicación del proyecto en actividades STEM de la universidad (Experiencia UVG, Curso de Vacaciones Mujeres en Ingeniería y Open House).

Este trabajo no aborda el diseño mecánico de nuevos actuadores ni la creación de hardware propietario. Asimismo, se reconocen limitaciones relacionadas con la latencia en algunos métodos de comunicación y con la resistencia mecánica de la mano animatrónica, las cuales quedan fuera del alcance de la presente investigación, pero se señalan como áreas de mejora en las recomendaciones.

De esta forma, el alcance de este proyecto se concentra en demostrar la viabilidad del Leap Motion como interfaz natural de control para distintos robots, integrando simulaciones, prototipos físicos y actividades educativas que fomentan la participación y el interés por la Ingeniería Mecatrónica.

CAPÍTULO 6

Marco teórico

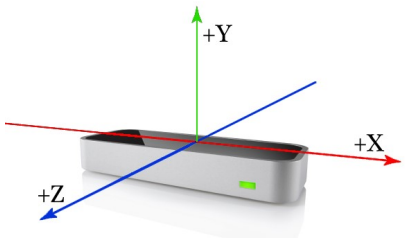
6.1. Leap Motion

El Leap Motion Controller (LMC) mostrado en la Figura 3 es un dispositivo de captura de movimiento que emplea dos cámaras y tres emisores de luz infrarroja (Figura 4) para detectar en tiempo real la posición de las manos y los dedos. El sensor genera imágenes en escala de grises a partir de la luz infrarroja reflejada, y mediante algoritmos de procesamiento reconstruye un modelo esquelético de la mano, permitiendo así el seguimiento preciso de sus movimientos [7].

El sensor LMC fue concebido principalmente para la detección de gestos de la mano y posiciones de los dedos en aplicaciones de software interactivo. En el estudio de Weichert, se desarrolló un entorno experimental utilizando un robot industrial KUKA KR 125/3 como referencia metrológica de alta precisión. Para ello, se fijó un lápiz en el extremo del robot y se desplazó su punta a posiciones conocidas dentro de un sistema de coordenadas. Estas posiciones fueron medidas simultáneamente por el LMC, lo que permitió comparar la ubicación real obtenida mediante la cinemática del robot con la estimada por el sensor [8].

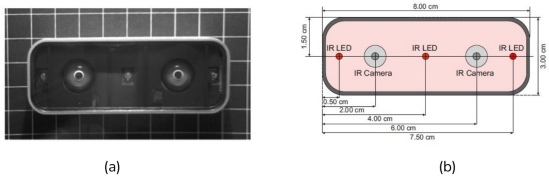
Si bien no fue posible alcanzar la precisión teórica de 0.01 mm declarada por el fabricante bajo condiciones de laboratorio, el desempeño del LMC se consideró altamente satisfactorio para su propósito principal en interfaces gestuales y aplicaciones interactivas, superando incluso a otros sensores de rango similar, como el Microsoft Kinect [8].

Figura 3. Leap Motion.



Nota. La imagen muestra el dispositivo Leap Motion junto a los ejes posibles de detectar. Esta imagen fue obtenida de [9].

Figura 4. Modelos de comunicación IPC.



Nota. La imagen muestra el dispositivo Leap Motion interno, mostrando sus cámaras y sensores infrarrojos. (a) Imágen real usando sensores infrarrojos (b) Esquemático del sensor. Esta imagen fue obtenida de [8].

6.2. Características y evolución de Leap Motion

Leap Motion fue fundada en 2010 por Michael Buckwald y David Holz, con el objetivo de desarrollar una interfaz natural (NUI, por sus siglas en inglés) que permitiera la captura precisa de los movimientos de la mano. Como resultado, se presentó el dispositivo denominado Leap Motion Controller, el cual alcanzaba una precisión superior a 0.01 mm. Su funcionamiento se basa en dos cámaras infrarrojas y tres emisores de luz infrarroja (LEDs IR), que proyectan un patrón de puntos en el espacio para detectar y reconstruir los movimientos de las manos en tres dimensiones [10].

En 2019, Leap Motion fue adquirida por la compañía Ultrahaptics, lo que dio lugar a la creación de *Ultraleap*. A partir de esta fusión, se consolidó un nuevo ecosistema de desarrollo que emplea el SDK *Gemini V5*, compatible con sistemas operativos Windows, Mac y Linux. Dicho SDK incorpora distintos modos de seguimiento: *Desktop*, diseñado para colocar el sensor de forma horizontal; *Head Mounted*, que permite su integración con dispositivos de realidad virtual; y *Screentop*, en el cual el sensor se coloca sobre la parte superior de una pantalla [11].

A lo largo de su evolución, el dispositivo ha enfrentado desafíos relacionados con la latencia residual, es decir, el retraso entre el movimiento real y la detección por el sistema. En sus primeras versiones se recomendaba el uso de puertos USB 3.0, monitores de baja latencia y la activación de modos como *High-Speed*, *Precision* y *Balanced*, según las necesi-

dades de la aplicación [10]. En contraste, las versiones más recientes (Leap Motion Orion y Ultraleap SDK) ajustan automáticamente dichos parámetros, ofreciendo opciones modernas: *Low Power Mode* (orientado al ahorro energético, equivalente al antiguo modo de precisión), *High Performance Mode* (máxima calidad de seguimiento, similar al antiguo modo de alta velocidad y precisión), *Light Robustness Mode* (optimización frente a condiciones lumínicas adversas) y *Fast Hands Mode* (mejor desempeño en movimientos rápidos)[12]. Pese a estas mejoras, se recomienda desactivar la opción *"Permitir que el equipo apague este dispositivo para ahorrar energía"* en los controladores USB Root Hub (3.0), a fin de garantizar una conexión estable y continua.

Entre las ventajas del sensor Leap Motion se encuentran su funcionamiento sin necesidad de contacto físico, su alta precisión en la detección de movimientos, su amplio campo de visión y la disponibilidad de un SDK que incluye ejemplos en Python.

6.3. Aplicaciones de Leap Motion

El sensor Leap Motion fue concebido esencialmente para permitir la interacción sin controladores físicos, lo que ha abierto un amplio abanico de aplicaciones. Entre estas se incluyen videojuegos, entretenimiento, arte digital, educación, investigación científica, procesos de rehabilitación, entornos de realidad virtual y aumentada, así como estrategias de publicidad interactiva.

En el ámbito de la robótica, existen múltiples trabajos que demuestran el potencial del Leap Motion como interfaz de control. Por ejemplo, el artículo *Natural gesture control of a delta robot using Leap Motion*, presentado por la School of Design de la South China University of Technology y el Guangdong Provincial Key Laboratory of Precision Equipment and Manufacturing Technology, describe un sistema implementado en Visual Studio 2010 utilizando C++ para controlar un robot delta [13].

Asimismo, en el artículo *Design of control system for articulated robot using Leap Motion sensor*, de la Shivaji University, se presenta el desarrollo de un sistema de control para un brazo robótico empleando Java y un microcontrolador Arduino [3].

6.4. Reconocimiento de gestos y comunicación

6.4.1. Distancia euclidiana

La distancia euclidiana se define como la medida de separación entre dos puntos en un espacio métrico, y corresponde a una generalización del teorema de Pitágoras a cualquier número de dimensiones. Su expresión general está dada por:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

donde x y y representan dos puntos en un espacio de n dimensiones [14].

- Para $n = 2$, la fórmula corresponde al teorema de Pitágoras en el plano cartesiano.
- Para $n = 3$, se obtiene su extensión al espacio tridimensional.
- Para valores mayores de n, se generaliza a espacios multidimensionales.

Es importante señalar que las coordenadas utilizadas deben estar expresadas en las mismas unidades de medida y definidas en ejes perpendiculares, de modo que la distancia calculada sea consistente.

De esta forma, el reconocimiento de gestos se implementó mediante el cálculo de la distancia euclidiana entre puntos característicos en el espacio 3D:

- V_1 representa el punto de referencia, la palma de la mano.
- V_2 representa la punta de un dedo.
- Las coordenadas están en un sistema cartesiano tridimensional (x, y, z).

La fórmula utilizada fue la siguiente:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Donde d es la distancia euclidiana entre el punto de la palma y el de cada dedo, permitiendo así determinar qué dedos están extendidos o cerrados.

6.4.2. Comunicación con Matlab, Robotis y Webots

Para el presente trabajo se ha utilizado Python 3.8, ya que es compatible con la versión Gemini V5 del SDK de Ultraleap. Este lenguaje permite recibir los datos del sensor Leap Motion y procesarlos en tiempo real.

Se emplearon distintos mecanismos de comunicación según la plataforma utilizada. En la interacción entre los procesos de Python y Matlab, se optó por el protocolo UDP debido a su baja latencia comparado con TCP, lo que lo hace adecuado para los experimentos en tiempo real del sistema de control gestual. Sin embargo, en otras partes del sistema, como la comunicación con los robots físicos o con plataformas de simulación, se utilizaron otros protocolos y métodos de intercambio de datos más apropiados.

Además, se implementó comunicación serial entre un proceso en Python y la placa Robotis OpenCM. Los gestos detectados por el sensor Leap Motion se transforman en cadenas de caracteres enviadas a través del puerto serial; la placa interpreta estas señales para accionar los motores correspondientes a cada gesto.

En el entorno del simulador Webots, la comunicación entre procesos se realizó inicialmente mediante el protocolo UDP para el envío de información desde Python hacia Matlab. Sin embargo, presentó un retraso notable, ya que entre la detección de un gesto y la reacción del robot simulado podían transcurrir aproximadamente 10 segundos.

6.4.3. Comunicación UDP

El *User Datagram Protocol* (UDP) es un protocolo de comunicación simple y orientado a datagramas, en el cual cada mensaje que la aplicación transmite corresponde a un Datagrama: [15] único. A diferencia de otros protocolos más complejos, UDP no ofrece retransmisión en caso de pérdida de datos, no garantiza la secuencia correcta de entrega, no elimina datagramas duplicados, ni implementa mecanismos de control de flujo o congestión. En consecuencia, no se adapta de manera automática a las condiciones variables del tráfico de red [16].

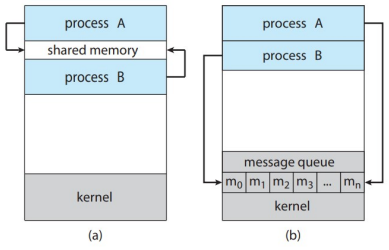
UDP incorpora un mecanismo de verificación denominado *checksum*, que permite detectar errores de transmisión en los datagramas. Este mecanismo es de tipo *end-to-end*, lo que significa que asegura que el mensaje recibido corresponda al que fue enviado originalmente. Sin embargo, en caso de que un datagrama esté dañado o se pierda durante la transmisión, el protocolo simplemente lo descarta sin reenviarlo. Para mitigar estas limitaciones, es posible implementar mecanismos adicionales en las aplicaciones, tales como reenvío de mensajes perdidos, numeración de secuencia o confirmaciones de entrega [16].

El protocolo UDP fue definido en la *RFC 768*, donde se establece como un protocolo sin conexión, orientado a mensajes y diseñado para aplicaciones que priorizan la baja latencia sobre la fiabilidad en la transmisión [17].

6.4.4. Comunicación entre procesos (IPC) y archivos temporales

En entornos donde se requiere cooperación entre procesos, resulta indispensable el uso de mecanismos de *Interprocess Communication* (IPC), que permiten el intercambio de datos entre aplicaciones independientes. Existen dos métodos principales: *Memoria compartida* (*shared memory*), donde los procesos acceden de manera conjunta a una misma región de memoria para leer y escribir datos, y *paso de mensajes* (*message passing*), mediante el cual los procesos intercambian información enviando y recibiendo mensajes de manera explícita como se muestra en la Figura 5 [18].

Figura 5. Modelos de comunicación IPC.



Nota. Esquema de modelos de comunicación. (a) Memoria compartida (b) Paso de mensajes. Esta imagen fue obtenida de [18].

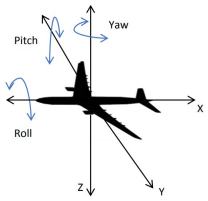
6.4.5. Ángulos de Euler

Los ángulos de Euler constituyen un método ampliamente utilizado para describir la orientación de un sistema de coordenadas respecto a otro en el espacio tridimensional. Esta representación se basa en la aplicación de tres rotaciones sucesivas alrededor de ejes principales, lo que permite parametrizar cualquier orientación mediante un conjunto de tres valores angulares (α, β, γ) [19].

Una de las parametrizaciones más empleadas en la literatura es la ZYZ Euler, en la cual la orientación se obtiene al realizar una primera rotación alrededor del eje z, seguida de una rotación alrededor del eje y, y finalmente otra rotación alrededor del eje z. Esta triple parametrización ofrece una descripción completa de la orientación espacial y resulta especialmente útil en aplicaciones de robótica y simulación [19].

Para comprender de manera más intuitiva los ángulos de Euler, puede hacerse una analogía con la trayectoria de un avión. Durante su vuelo, la aeronave realiza combinaciones de giros en torno a los tres ejes principales: X, y y z [20]. El movimiento alrededor del eje longitudinal (x) se denomina alabeo o *roll*. El movimiento alrededor del eje transversal (y) corresponde a la elevación o *pitch*.El movimiento alrededor del eje vertical (z) se conoce como cabeceo o *yaw* [20]. Estos tres giros permiten describir de forma completa la orientación del avión en el espacio. Dichos giros se ilustran en la Figura 6.

Figura 6. Ángulos de Euler.



Nota. La imagen muestra la *trayectoria de un avión* alrededor de los ejes x, y y z. Esta imagen fue obtenida de [20].

CAPÍTULO 7

Entorno de desarrollo y configuraciones iniciales

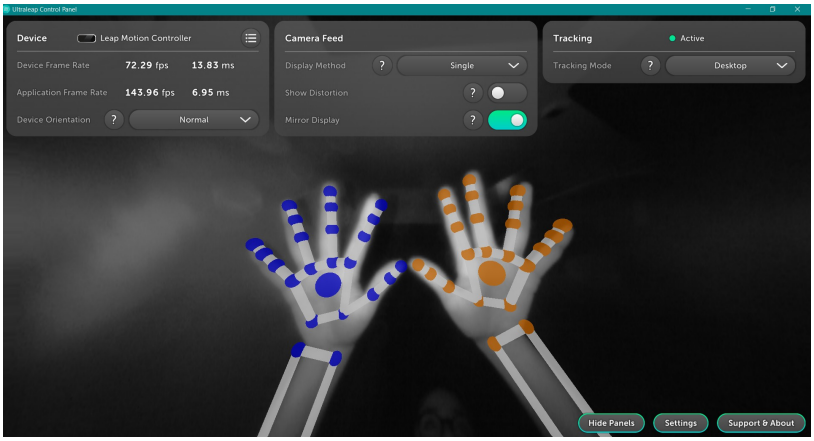
En este capítulo se presentan los elementos fundamentales necesarios para la preparación del entorno de trabajo previo a la programación e implementación del proyecto. Se describen tanto los recursos de hardware como de software empleados, junto con las configuraciones iniciales requeridas para garantizar un funcionamiento adecuado del sistema.

7.1. Leap Motion

Para iniciar el desarrollo se utilizó el sensor Leap Motion Controller 1, fabricado originalmente por la empresa Leap Motion y actualmente comercializado por Ultraleap. Con el cambio de propietario, el *Software Development Kit* (SDK) también migró y sufrió diversas modificaciones. En la fase inicial se llevó a cabo una búsqueda del SDK original. No obstante, la adquisición de la empresa desarrolladora dificultó la localización de una versión disponible o adecuadamente documentada. Se encontró una versión en *GitHub*, aunque no funcionó de manera adecuada debido a incompatibilidades con las nuevas actualizaciones de Windows. Finalmente, se optó por utilizar el *Gemini V5 SDK* de Ultraleap, el cual es una versión más reciente, a pesar de que el sensor había sido creado bajo otra compañía.

Al conectar el sensor a la computadora, se puede acceder al programa *Ultraleap Tracking*, el cual incluye el *Ultraleap Control Panel* como se muestra en la Figura 7. Este panel visualiza el seguimiento de las manos detectadas y ofrece diversa información relacionada con el dispositivo, las cámaras y el proceso de rastreo.

Figura 7. Panel de control Leap Motion.



Nota. La imagen muestra panel de control de Leap Motion, mostrando el seguimiento de las manos. Elaboración propia.

En la sección correspondiente al dispositivo se muestran indicadores como *Device Frame Rate*, *Application Frame Rate* y *Device Orientation*. Particularmente, este último resulta de gran relevancia, ya que permite elegir entre las opciones *Normal* e *Inverted*, determinando la orientación del sensor como se muestra en la Figura 8, en función de la posición física que se le asigne para detectar las manos de manera adecuada.

Figura 8. Leap Motion (orientación).

Nota. La imagen muestra el dispositivo *Leap Motion* en distintas orientaciones.(a) Normal (b) Inverted. Elaboración propia.

En la sección de *Camera Feed* se encuentra el parámetro *Display Method*, el cual ofrece las opciones *Single* y *Side by Side*. Estas permiten elegir si la imagen capturada por la cámara se muestra una sola vez o duplicada en paralelo. Adicionalmente, puede activarse la opción *Show Distortion*, que permite visualizar la imagen considerando la corrección de distorsión para otras dimensiones, así como la opción *Mirror Display*, que aplica un efecto de espejo a la visualización.

Finalmente, en el apartado de *Tracking* se incluyen tres configuraciones fundamentales: *Desktop*, *Head Mounted* y *ScreenTop*. Según la documentación de Ultraleap [21], cada una de estas opciones corresponde a una orientación específica del sensor:

- *Desktop*: El dispositivo debe colocarse sobre una mesa, apuntando hacia arriba.
- *Head Mounted*: El sensor se monta en un casco de realidad virtual o aumentada (VR/AR).
- *ScreenTop*: El dispositivo se sitúa sobre una pantalla, apuntando hacia abajo en dirección al usuario.

7.2. Reconocimiento de gestos y comunicación

En el contexto de este proyecto de graduación, se empleó la distancia euclidiana en tres dimensiones para calcular la separación entre puntos en el espacio cartesiano tridimensional. Esta propiedad resulta especialmente útil porque la medida es invariante ante rotaciones, lo que significa que, independientemente de la orientación de la mano, la distancia relativa entre los dedos permanece constante [22].

El reconocimiento de gestos mediante el sensor Leap Motion se basó en el cálculo de la distancia euclidiana entre puntos en el espacio tridimensional, complementado con el análisis de los ángulos de orientación *roll*, *pitch* y *yaw*. Estos parámetros permiten identificar con mayor precisión la posición y el movimiento de la mano, facilitando la detección de gestos específicos.

Los ángulos de Euler se utilizaron para el movimiento de la mano simulada en Matlab. En este caso

- ***Roll:*** Describe el giro horario o antihorario de la mano.
- ***Pitch:*** Corresponde a los movimientos de flexión y extensión.
- ***Yaw:*** Se asocia a los movimientos de abducción y aducción de los dedos.

El uso de esta parametrización resulta intuitivo, pues se relaciona directamente con los grados de libertad más relevantes en los gestos humanos, facilitando así su aplicación en la simulación de movimientos naturales.

La comunicación entre los diferentes módulos del sistema se implementó utilizando tres enfoques principales: El protocolo UDP, la comunicación serial y el uso de archivos temporales. Cada uno de estos mecanismos presentó ventajas y limitaciones particulares, lo que motivó su análisis y aplicación en distintas fases del desarrollo.

7.3. Entornos y plataformas de desarrollo

En el desarrollo del proyecto se emplearon diversas plataformas de software que facilitaron tanto la programación como la simulación y el control de los sistemas robóticos. Entre las principales se encuentran Python, Matlab, Webots y Robotis. Los distintos códigos de estas plataformas se pueden encontrar en el Anexo 16.5. Durante el desarrollo del proyecto,

se utilizó ChatGPT [23] como apoyo para la depuración de fragmentos de código en Robotis, Webots, Python y Matlab, lo que permitió agilizar la implementación de algoritmos de control y comunicación. De esta manera, la integración de estas plataformas permitió una transición fluida entre la simulación virtual y la implementación práctica, asegurando mayor confiabilidad en los resultados obtenidos.

7.3.1. Python

El sensor Leap Motion fue programado utilizando el lenguaje Python, dado que el SDK de Ultraleap proporciona una amplia variedad de ejemplos y librerías compatibles para este entorno. La programación se realizó específicamente en la versión 3.8.0 de Python, dentro de un entorno virtual configurado para este proyecto de graduación. La utilización de un entorno virtual permitió instalar y administrar las librerías necesarias de forma aislada, sin afectar otras configuraciones del sistema.

La elección de esta versión particular de Python respondió a su compatibilidad con Matlab, lo cual facilitó la integración de ambas plataformas durante las etapas de simulación y validación.

Los principales paquetes instalados en dicho entorno virtual, se muestran en el Anexo 16.1 los cuales resultaron esenciales para el desarrollo del proyecto.

7.3.2. Matlab

Para el desarrollo del proyecto se utilizó Matlab R2022a, el cual permitió realizar diversas simulaciones orientadas a la interpretación de gestos captados por el Leap Motion desde Python. De esta forma, fue posible establecer un puente entre el reconocimiento de movimientos de la mano y la ejecución de acciones sobre diferentes sistemas.

Adicionalmente, Matlab se integró con el simulador Webots, lo cual facilitó la recreación de entornos virtuales para la validación de estrategias de control en robots móviles y manipuladores. Este entorno de simulación resultó esencial para evaluar el comportamiento de los algoritmos antes de llevarlos a la implementación física, reduciendo riesgos y asegurando un mejor aprovechamiento de recursos en la fase experimental. La combinación de Matlab y Webots permitió una transición progresiva desde la simulación hasta el prototipado real, manteniendo coherencia entre los resultados obtenidos en ambos entornos.

7.3.3. Robotis

La plataforma Robotis fue empleada para la programación y control del microcontrolador *OpenCM9.04 C*, encargado de accionar la mano animatrónica utilizada en el proyecto. Este sistema consta de un total de seis servomotores de la serie *Dynamixel XL-320*, destinados al control de los dedos, mientras que la muñeca y el antebrazo se implementaron con servomotores *Dynamixel AX-12A*, seleccionados por su mayor torque y robustez mecánica [24].

Un aspecto relevante es que se disponía de una programación base desarrollada en años anteriores, la cual sirvió como punto de partida para la implementación actual. Sobre esta base se realizaron ajustes y mejoras para adecuarla a los nuevos requerimientos del proyecto, optimizando la comunicación, el mapeo de gestos y la coordinación de los actuadores. La elección de la plataforma Robotis no sólo permitió aprovechar hardware confiable, sino que también facilitó la integración con las demás herramientas empleadas, como Python, generando un ecosistema de desarrollo flexible.

CAPÍTULO 8

Interfaz de selección y control basado en agente puntual

8.1. Interfaz gráfica de selección de sistema robótico

Se desarrolló una interfaz gráfica en Python, ilustrada en la Figura 9, cuyo propósito es ofrecer al usuario un medio intuitivo para seleccionar el sistema robótico que desea utilizar. La aplicación permite elegir entre seis opciones principales:

- Agente puntual.
- Pololu físico.
- Pololu simulado.
- Mano animatrónica.
- Mano virtual.
- Robot Sawyer simulado.

Las opciones se presentan en una ventana organizada en dos filas de tres elementos, cada uno representado mediante una imagen identificativa del sistema correspondiente. De esta forma, el diseño visual facilita la interacción y permite que el usuario reconozca rápidamente el robot o simulación a ejecutar.

La funcionalidad principal de la interfaz consiste en que, al hacer clic sobre cualquiera de las imágenes, el usuario es dirigido al módulo correspondiente donde se muestran las instrucciones del sistema seleccionado. En el caso del Pololu, el clic lleva a un submódulo adicional en el que el usuario puede elegir el modo de juego que desea utilizar, tal como se

muestra en el Anexo 16.2. Una vez revisadas las instrucciones, al presionar el botón Listo, la interfaz ejecuta automáticamente el *script* asociado al sistema o modo de juego elegido.

Cada *script* se ejecuta en un proceso independiente, lo que garantiza que la interfaz principal continúe activa y disponible, sin bloquearse durante la operación del sistema seleccionado. Este enfoque modular y visual no solo simplifica la experiencia del usuario, sino que también permite integrar en una misma plataforma todos los sistemas desarrollados durante el proyecto, brindando un acceso centralizado y eficiente a cada uno de ellos.

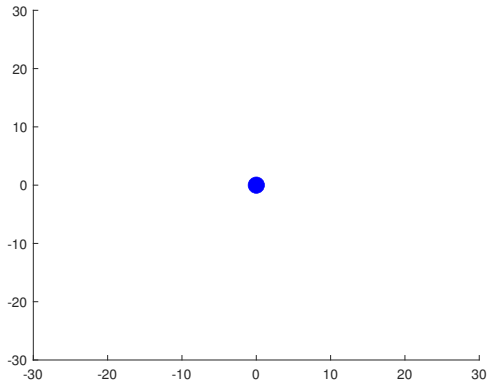
Figura 9. Interfaz gráfica.

Nota. La imagen muestra la interfaz en Python para seleccionar el sistema robótico a utilizar. Elaboración propia.

8.2. Agente puntual

La primera simulación desarrollada correspondió a un agente puntual, mostrada en la Figura 10. El propósito de esta implementación inicial fue contar con un modelo experimental sencillo que permitiera validar, de manera preliminar, la lógica de control basada en gestos antes de aplicarla directamente al robot Pololu 3Pi+.

Figura 10. Agente puntual en Matlab.



Nota. La imagen muestra el agente puntal simulado en Matlab. Elaboración propia.

El agente puntual simulado se utilizó como una representación conceptual del robot, donde los gestos captados por el sensor Leap Motion se tradujeron en movimientos básicos del agente puntual dentro del entorno de simulación. Este enfoque permitió comprobar la relación entre los gestos definidos y las acciones de desplazamiento (avance, retroceso, giros y detención), asegurando que el mapeo de gestos estuviera correctamente implementado.

De esta manera, la simulación del agente puntual funcionó como un banco de pruebas inicial, sirviendo para depurar la comunicación entre los procesos en Python y Matlab, así como para identificar posibles problemas de latencia o de interpretación de gestos. Gracias a este primer modelo, fue posible refinar la metodología de control antes de integrarla en simulaciones más complejas con el Pololu 3Pi+ en Webots.

8.2.1. Control de agente puntual en Matlab

El *script* desarrollado en Matlab implementa una interfaz gráfica que simula un *joystick* virtual controlado mediante los gestos de la mano derecha. De esta forma, la esfera puede desplazarse a lo largo de los ejes x y y, o bien realizar movimientos combinados que permiten un control de 360 grados dentro del entorno de simulación.

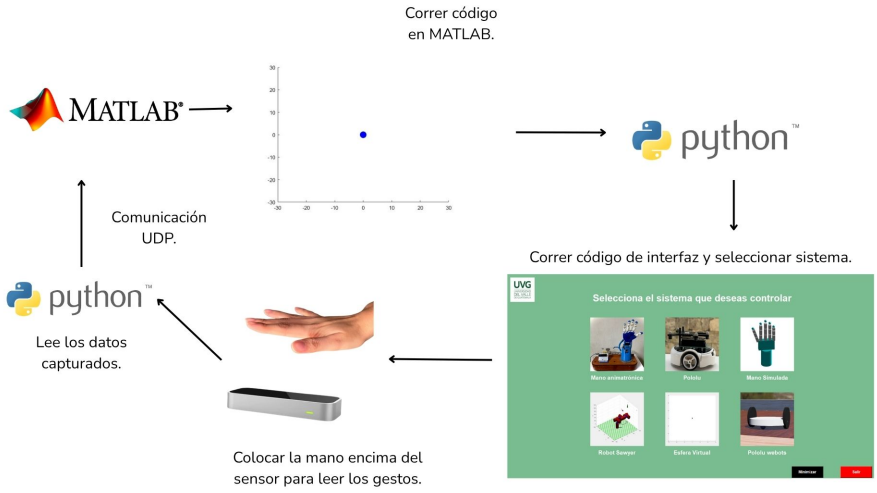
Como medida de seguridad, se incluyó un sistema de freno de emergencia activado con la mano izquierda, el cual detiene de inmediato el movimiento de la esfera al ser detectado. Además, la interfaz gráfica cuenta con un indicador textual ubicado en el interior de la figura, que muestra en tiempo real la acción correspondiente a cada gesto. Este elemento visual permitió mejorar la interacción usuario-sistema, ofreciendo retroalimentación inmediata sobre los comandos ejecutados.

8.2.2. Control de agente puntual en Python

El lenguaje Python se empleó para la captura de los gestos detectados por el sensor Leap Motion, los cuales son posteriormente enviados a Matlab mediante el protocolo UDP. En este esquema de comunicación, cada gesto se encapsula en un datagrama que incluía en su cabecera el número de puerto de transporte como identificador del canal de comunicación entre procesos. De esta manera, el programa en Python actua como emisor de los gestos, mientras que en Matlab funciona como receptor, procesando la información recibida y traduciéndola en movimientos de la esfera.

El datagrama es dirigido a la dirección IP previamente configurada, asegurando que los comandos lleguen de forma correcta y sin ambigüedad al sistema de simulación. Este mecanismo permitió mantener una interacción en tiempo real entre los gestos humanos y la respuesta del modelo virtual, como se ilustra en la Figura 11.

Figura 11. Diagrama de comunicación de esfera virtual.



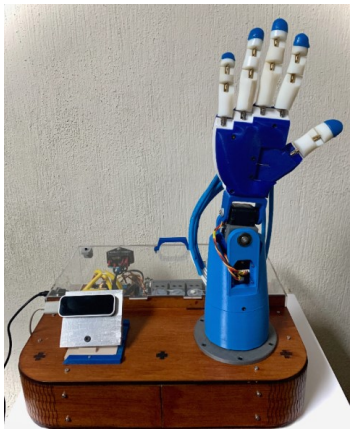
Nota. La imagen muestra el diagrama del agente puntual simulado en Matlab mostrando su comunicación entre plataformas. Elaboración Propia.

CAPÍTULO 9

Control de la mano robótica animatrónica y simulada

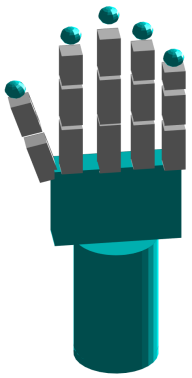
Se empleó una mano animatrónica previamente desarrollada en la Universidad del Valle de Guatemala [25] y, adicionalmente, se diseñó una mano simulada en Matlab (Figura 13). El objetivo principal de ambos modelos fue replicar en tiempo real los gestos captados por el sensor Leap Motion, específicamente aquellos realizados con la mano derecha.

Figura 12. Mano animatrónica.



Nota. La imagen muestra la mano animatrónica abierta. Esta imagen fue obtenida de [25].

Figura 13. Mano simulada.



Nota. La imagen muestra la mano simulada abierta. Elaboración propia.

La mano animatrónica constituye un sistema mecatrónico diseñado para emular los movimientos básicos de la mano humana. Su estructura está conformada por falanges articuladas y accionadas mediante servomotores, los cuales reproducen el movimiento de los dedos en respuesta a señales de control externas [24].

En el marco de este proyecto, la mano animatrónica y simulada se configuraron para reconocer y reproducir tres tipos de gestos fundamentales:

- **Mano abierta:** Todos los dedos extendidos, representando la posición de reposo natural al no aplicar fuerza de cierre.
- **Mano cerrada:** Flexión simultánea de todos los dedos, simulando el gesto de puño.
- **Movimientos de dedos por separado:** Control individual de cada dedo, lo cual permite un mayor grado de expresividad y la posibilidad de ejecutar gestos más complejos.

La integración de la mano animatrónica y simulada con el sensor Leap Motion se realizó con el fin de demostrar la capacidad del sistema para interpretar gestos humanos y transformarlos en movimientos mecánicos equivalentes.

9.1. Mano animatrónica y entornos de desarrollo

Para la implementación de la mano animatrónica se emplearon dos plataformas principales: Python y Robotis. La primera se utilizó como medio de comunicación con el sensor Leap Motion, permitiendo interpretar los gestos captados y transformarlos en señales de control. La segunda se empleó para la programación y gestión del microcontrolador *OpenCM9.04 C*, encargado de accionar los servomotores *Dynamixel* instalados en la mano.

En las primeras pruebas, el sistema funcionó correctamente, logrando replicar los gestos solicitados mano abierta, mano cerrada y movimientos de dedos por separado de forma

fluida. No obstante, tras varios ensayos, la mano dejó de ejecutar los gestos completos; únicamente se observaba movimiento en los servomotores, pero sin que estos generaran la respuesta mecánica esperada en los dedos.

Este comportamiento llevó a la conclusión de que ocurrió un desperfecto mecánico en los hilos de tensión que transmiten el movimiento de los servomotores hacia las poleas que accionan los dedos. A pesar de esta limitación, la experiencia permitió validar la comunicación entre el sensor y la plataforma de control, dejando en evidencia que el principal desafío radica en la parte mecánica del sistema, más que en la programación o la electrónica.

9.1.1. Control de la mano animatrónica mediante Python

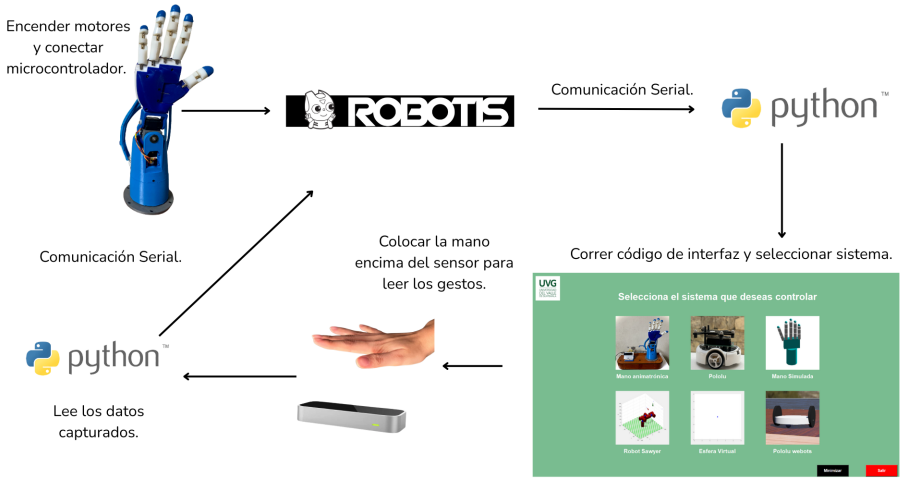
Dado que el sensor Leap Motion se programa en Python, se desarrolló una aplicación capaz de captar los gestos detectados y procesarlos mediante el cálculo de la distancia euclidiana entre la palma y la punta de los dedos. Con este método fue posible determinar la posición de la mano en cada instante y clasificar el gesto correspondiente.

Una vez identificados, los gestos eran enviados al monitor serial de Robotis en forma de un arreglo de 10 bytes. En las primeras pruebas, el sistema asignaba una variable a cada gesto y enviaba únicamente ese valor al monitor serial para verificar de manera rápida si la mano ejecutaba correctamente los movimientos de los motores.

Posteriormente, se optó por enviar directamente el paquete completo de bytes, lo que permitió transmitir la información de manera más precisa y confiable, asegurando que cada articulación recibiera el valor correspondiente dentro de la estructura del paquete.

Debido a que la comunicación se establecía a través del puerto serial, fue necesario garantizar que el usuario seleccionara siempre el puerto correcto antes de iniciar la ejecución del sistema. La lógica de interacción implementada en este módulo se resume en la Figura 14.

Figura 14. Diagrama de interacción entre hardware, software e interfaz.



Nota. La imagen muestra el *diagrama de interacción* entre el sensor Leap Motion, la interfaz en Python y su comunicación. Elaboración propia.

9.1.2. Control de la mano animatrónica mediante Robotis

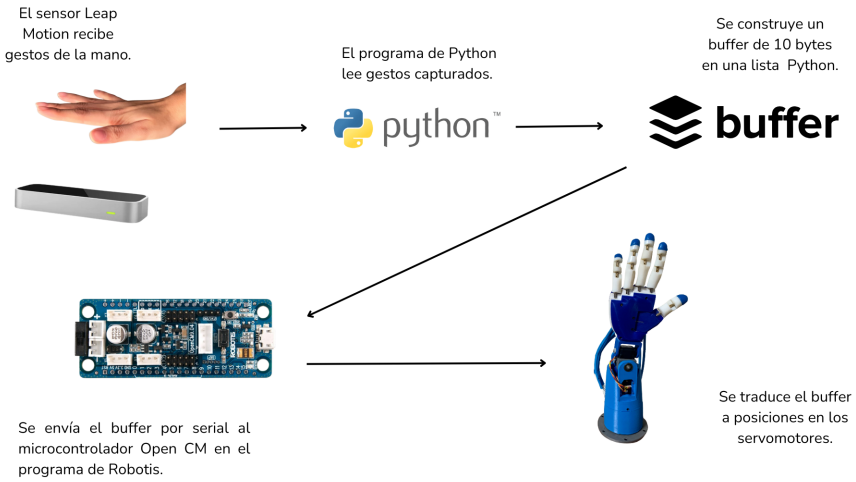
La programación de los motores de la mano animatrónica se realizó utilizando la plataforma Robotis. Este módulo recibía los datos enviados desde Python por medio del puerto serial y, en función de ellos, ejecutaba el movimiento correspondiente en la mano. Un aspecto relevante fue el orden de ejecución de los programas: El código en Robotis debía ser ejecutado antes que el código en Python, de lo contrario se producían interferencias en la comunicación serial. Los datos recibidos correspondían a un paquete serial compuesto por 10 bytes, cada uno con una función específica dentro del control de la mano animatrónica. La estructura del paquete era la siguiente:

$$[255,\ ante,\ mune\_des,\ mune\_ext,\ pulg,\ pulg\_meta,\ indi,\ medi,\ anul,\ meni]$$

El primer byte (255) actuaba como identificador de inicio. Los siguientes tres bytes codificaban las posiciones del antebrazo y de las dos articulaciones de la muñeca. Finalmente, los seis bytes restantes correspondían a los dedos, los cuales utilizaban una codificación binaria: El valor `0` representaba un dedo cerrado, mientras que cualquier valor distinto de cero indicaba un dedo abierto.

De esta forma, cada dedo de la mano animatrónica podía ser controlado individualmente en función de los gestos detectados por el Leap Motion. La lógica de comunicación implementada para este sistema se muestra en la Figura 15.

Figura 15. Diagrama de flujo de mano animatrónica Robotis.



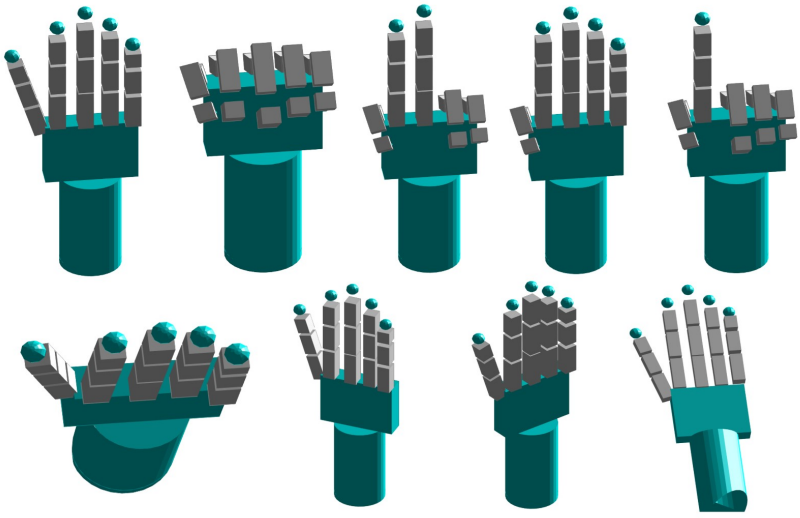
Nota. La imagen muestra el diagrama de flujo de la programación de la mano animatrónica en Robotis. Elaboración propia.

9.2. Mano simulada y entornos de desarrollo

Se implementó una mano simulada en Matlab, cuyo propósito fue replicar tipo espejo los gestos captados por el sensor Leap Motion y procesados en Python. Este modelo virtual se observa en la Figura 16. Permitió visualizar de manera inmediata la respuesta del sistema a los diferentes gestos realizados, sin depender exclusivamente de la mano animatrónica física.

La comunicación entre los programas desarrollados se estableció mediante el protocolo UDP, el cual facilitó la transmisión de datos en tiempo real con baja latencia. Gracias a este enfoque, los gestos detectados con el programa de Python eran enviados como paquetes de datos hacia el programa de Matlab, donde se interpreta la información y se actualizaba el estado de la mano simulada. Este esquema de comunicación permitió mantener una conexión continua y eficiente entre ambas plataformas.

Figura 16. Mano simulada en distintas posiciones.



Nota. La imagen muestra la mano simulada en Matlab en distitinas posiciones. Elaboración propia.

9.2.1. Control de la mano simulada en Matlab

El diseño de la mano simulada se desarrolló en Matlab, construyendo cada una de sus partes de manera modular. La palma se generó mediante la función `drawCuboid`, que emplea la instrucción `patch` para formar un paralelepípedo de las dimensiones especificadas. El antebrazo, por su parte, se modeló utilizando `cylinder` y `surf`, extruyendo un cilindro hacia abajo y cerrando sus extremos con dos discos. Ambos elementos fueron agrupados en un objeto `hgtransform`, lo que permitió manipularlos en conjunto como si se tratara de una sola pieza.

Cada dedo estuvo compuesto por tres falanges representadas con cubos alargados y una punta esférica. La jerarquía entre falanges se definió de la siguiente manera:

- **Proximal (prox):** Base del dedo, conectada directamente a la palma.
- **Medial (med):** Hijo del proximal, rotado y trasladado en relación a este.
- **Distal (dist):** Hijo del medial, rotado y trasladado, arrastrado por los movimientos de las falanges anteriores.

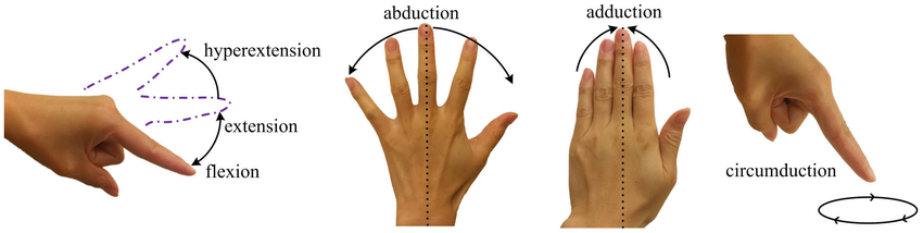
Con esta estructura jerárquica, cuando la falange proximal realiza una rotación, arrastra consigo a la medial y a la distal, emulando de manera realista el comportamiento de un dedo humano.

Para el diseño y control de los movimientos se emplearon los ángulos de Euler (*roll*, *pitch* y *yaw*):

- **Roll:** Controla la dirección lateral de la mano (izquierda y derecha).
- **Pitch:** Define la inclinación en el plano sagital, es decir, la flexión y extensión de la muñeca; en *Matlab* se implementa como una rotación alrededor del eje x.
- **Yaw:** Asociado a la abducción y aducción, se empleó principalmente en el diseño para determinar la separación o cercanía de los dedos.

Para una mejor comprensión del efecto de estos ángulos en la simulación, se presenta la Figura 17.

Figura 17. Movimientos de la mano.



Nota. La imagen muestra los disntintos movimientos de la mano humana. Esta imagen fue obtenida de [26].

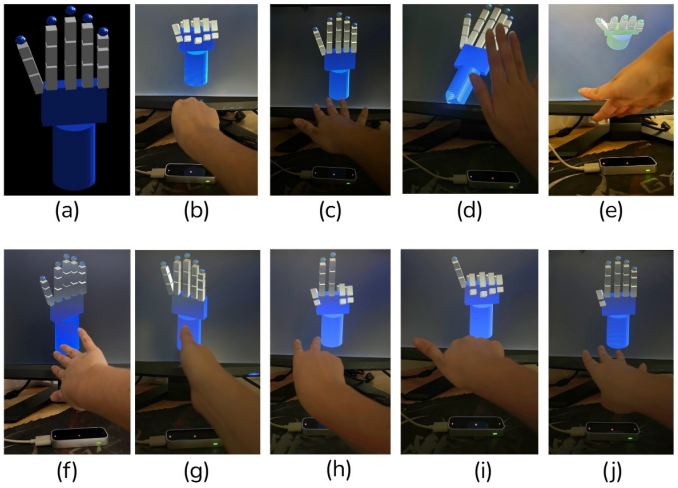
9.2.2. Control de la mano simulada en Python

En Python se implementó la lógica de detección y procesamiento de gestos captados por el sensor Leap Motion. Para determinar el estado de la mano (abierta, cerrada o con movimientos individuales de los dedos) se empleó el cálculo de la distancia euclidiana entre la palma y las puntas de los dedos. De esta manera fue posible identificar de forma precisa qué dedos se encontraban extendidos y cuáles en flexión.

Adicionalmente, se calcularon los ángulos de Euler necesarios para describir la orientación de la mano. En particular, se utilizó el ángulo *roll* para controlar la dirección lateral (izquierda y derecha) y el ángulo *pitch* para representar movimientos de flexión y extensión en el eje adelante–atrás.

La integración entre Python y Matlab permitió que los gestos detectados por el sensor fueran enviados y replicados en la mano simulada, tal como se ilustra en la Figura 18.

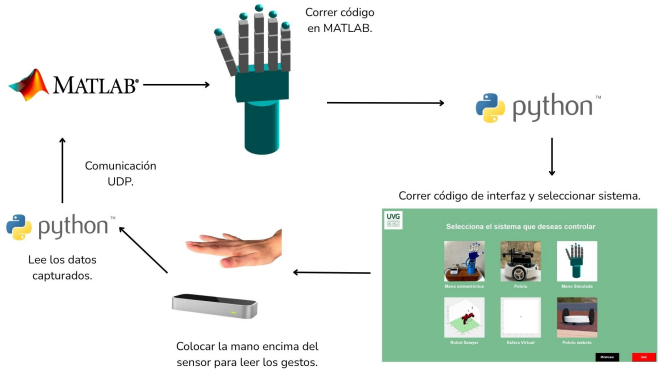
Figura 18. Mano simulada con Leap Motion.



Nota. La imagen muestra la mano simulada en Matlab con un efecto tipo espejo en distitinas posiciones con vista a la utilización del sensor Leap Motion.(a) Simulación (b) Mano cerrada, (c) Mano abierta, (d) Extensión, (e)Flexión, (f) Mano abierta antihorario, (g) Mano abierta horario, (h) Índice y medio, (i) Pulgar, (j) Pulgar abajo. Elaboración propia.

Finalmente, la lógica de interacción y comunicación de este sistema se resume en la Figura 19.

Figura 19. Diagrama de mano simulada con Leap Motion.



Nota. La imagen muestra el diagrama de la *mano simulada* en Matlab y Python asi como su interacción y comunicación. Elaboración propia.

CAPÍTULO 10

Control Pololu 3Pi+ simulado y físico

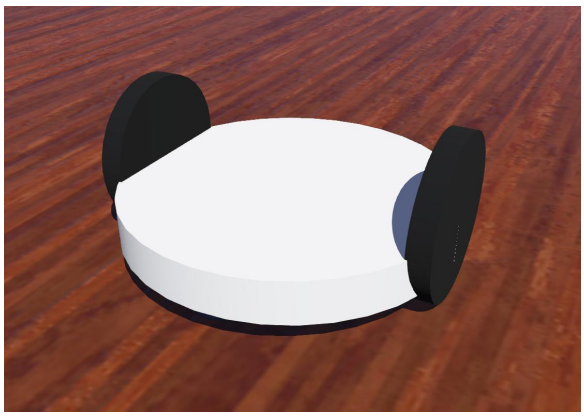
El robot Pololu 3Pi+ fue simulado en el entorno Webots (Figura 20), recibiendo como entradas los gestos captados por el sensor Leap Motion y procesados por Python mediante el mismo método de la distancia euclidiana. De esta forma, los movimientos de la mano del usuario se tradujeron en acciones sobre el modelo virtual del robot.

La comunicación inicial entre los programas de Python y Webots se estableció a través del protocolo UDP. No obstante, se observó un retraso considerable (*lag*) en la transmisión; al momento en que Webots recibía un gesto, la respuesta no era inmediata, lo que afectaba la interacción en tiempo real. Como alternativa, se probó el uso de *Pipes:* [27] para acelerar la comunicación. Si bien esta técnica ofreció una mejora en la velocidad, presentó errores frecuentes de bloqueo cuando Matlab y Python no se ejecutaban de forma sincronizada. En dichos casos, el canal de comunicación quedaba bloqueado y Webots arrojaba errores debido a la limitada compatibilidad de Matlab con *pipes* en sistemas Windows. Estos inconvenientes provocaban que el controlador de Webots se detuviera o no pudiera establecer conexión con Python, lo que llevó a descartar esta opción.

Finalmente, se adoptó un canal de comunicación más simple y robusto basado en el uso de archivos temporales. Esta técnica, propia de la comunicación entre procesos (IPC), consiste en que un proceso escribe la información en un archivo que otro proceso lee posteriormente. Su principal ventaja radica en la amplia compatibilidad entre plataformas y lenguajes de programación, lo que permitió integrar de manera más confiable los módulos del sistema [28].

Aunque los archivos temporales presentan mayor latencia en comparación con métodos como la memoria compartida, demostraron ser más estables que alternativas como UDP o *pipes*. Además, cuentan con la característica de almacenarse en directorios especiales del sistema y eliminarse automáticamente al finalizar el programa o al reiniciar el sistema operativo, evitando la acumulación de datos y garantizando una gestión eficiente [28].

Figura 20. Pololu simulado en Webots.



Nota. Elaboración propia.

El robot Pololu 3Pi+ fue utilizado en su versión física, como se muestra en la Figura 21. Para lograr esta integración se empleó una combinación con el sistema del agente puntual, equivalente a un *joystick*, que replicaba el mismo esquema de movimientos definidos en la simulación de Webots. De esta manera, se mantuvo la coherencia en la lógica de control, utilizando nuevamente el protocolo UDP para la comunicación entre plataformas.

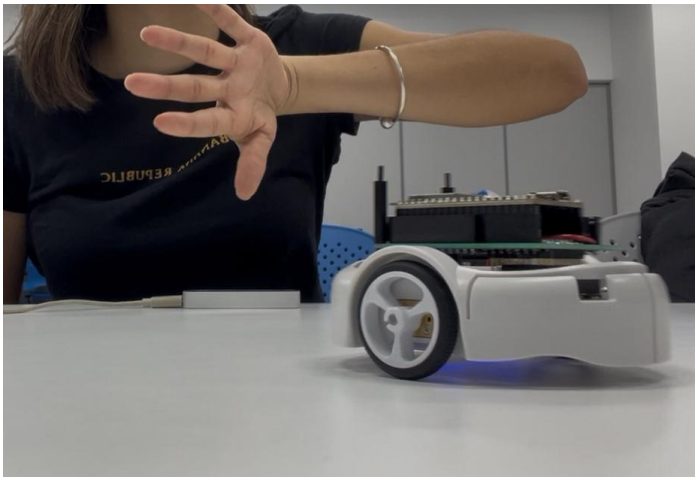
El sistema logró funcionar adecuadamente. Sin embargo, al igual que en la simulación en Webots, se presentó un retraso significativo en la respuesta del robot, alcanzando aproximadamente los 10 segundos entre la detección del gesto y la ejecución de la acción correspondiente.

Tras un análisis más detallado, se determinó que la causa principal del retardo estaba en la cantidad de comunicaciones intermedias requeridas por el sistema. El flujo de información seguía la siguiente secuencia: El sensor capturaba los gestos, estos eran enviados al programa en Python, posteriormente transmitidos por UDP al programa de Matlab, luego reenviados por TCP al robot, y finalmente este ejecutaba la instrucción. Cada una de estas capas de comunicación añadía tiempo de procesamiento y transmisión, lo que generaba acumulación de retardos.

Adicionalmente, como señala Stevens [16], los mecanismos inherentes al protocolo TCP como el control de flujo y la retransmisión de paquetes pueden introducir retardos adicionales en aplicaciones interactivas. Por esta razón, se concluyó que la arquitectura debía simplificarse, optando por utilizar únicamente Python como plataforma de comunicación directa con el robot, dado que es también la misma herramienta utilizada para programar el sensor Leap Motion.

Con esta modificación, el sistema redujo la complejidad de la comunicación y mejoró la capacidad de respuesta en tiempo real, favoreciendo la interacción natural entre el usuario y el robot físico.

Figura 21. Robot Pololu 3Pi+ físico.



Nota. La imagen muestra el robot Pololu 3Pi+ físico siendo controlado por el sensor Leap Motion. Elaboración propia.

10.1. Control de simulación del robot Pololu 3Pi+ en Webots

En el simulador Webots, el robot Pololu 3Pi+ fue definido dentro de un archivo de mundo, el cual describe tanto su estructura geométrica como sus características físicas.

Para complementar la simulación, se incorporaron sensores virtuales tales como el *GPS* y la brújula, que permiten obtener en tiempo real la posición absoluta del robot y su orientación dentro del espacio simulado. La información de estos dispositivos resulta fundamental para la implementación de algoritmos de navegación y control, ya que provee retroalimentación directa sobre el estado del sistema.

En el caso particular de la simulación en Webots, se optó por el método de paso de mensajes debido a la heterogeneidad de las plataformas involucradas: Python, encargado del procesamiento de gestos provenientes del Leap Motion, y Matlab/Webots, encargado de la simulación del robot Pololu. Dado que estas aplicaciones no comparten el mismo espacio de direcciones, la implementación de memoria compartida no era viable [18].

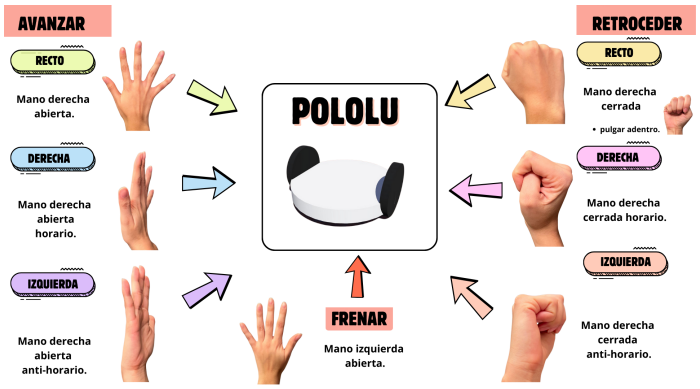
Para establecer la comunicación, se recurrió al uso de archivos temporales, lo que permitió una integración sencilla y robusta entre los procesos. Aunque esta técnica no garantiza la menor latencia posible, estudios realizados por el MIT Lincoln Laboratory demuestran que los esquemas de comunicación basados en archivos pueden escalar de manera eficiente en sistemas de alto rendimiento, lo cual respalda su aplicación en la presente simulación [28].

10.2. Control de simulación del robot Pololu 3Pi+ en Python

En la simulación del robot Pololu 3Pi+, el lenguaje Python se empleó como principal para la lectura y procesamiento de los gestos detectados por el sensor Leap Motion. El objetivo fue establecer un sistema de control gestual en el que los movimientos de la mano del usuario se tradujeran directamente en comandos de navegación para el robot dentro del entorno de Webots.

El esquema de control se definió como se muestra en la Figura 22

Figura 22. Gestos del robot Pololu 3Pi+ simulado.



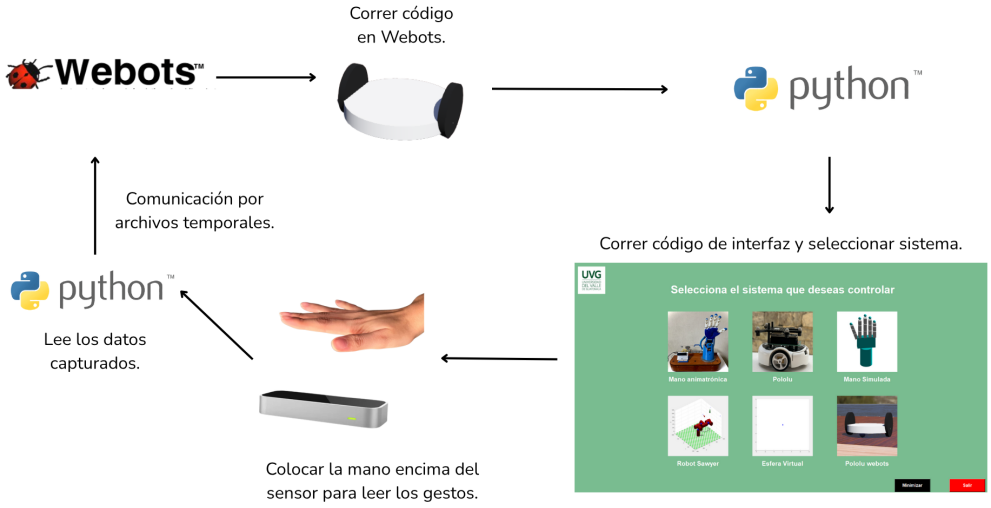
Nota. Elaboración propia.

Con este sistema, la mano derecha controla los desplazamientos del robot en distintas direcciones, mientras que la mano izquierda cumple la función de seguridad, permitiendo detener el avance del robot en cualquier momento.

La comunicación entre los programas de Python y Webots se implementó mediante el uso de archivos temporales. En este esquema, el programa de Python se encarga de capturar los gestos del Leap Motion y escribirlos en un archivo temporal, mientras que Webots lee dicho archivo en cada ciclo de simulación para actualizar el control del robot.

El empleo de archivos temporales presentó varias ventajas en comparación con otros métodos probados (como UDP o *pipes*). Si bien este enfoque introduce una latencia ligeramente mayor que otros métodos de comunicación, resultó ser la alternativa más estable y confiable para mantener la interacción entre el sensor, Python y el simulador Webots. De esta manera, se garantizó que el robot respondiera a los gestos del usuario de forma coherente y continua durante toda la simulación. La lógica de la interacción y comunicación entre Matlab, Python y el simulador Webots se resume en la Figura 23.

Figura 23. Diagrama de mano simulada con Leap Motion.



Nota. La imagen muestra el diagrama del robot Pololu 3Pi+ en los programas de Webots y Python así como su interacción y comunicación. Elaboración propia.

10.3. Control de Pololu 3Pi+ físico en Python

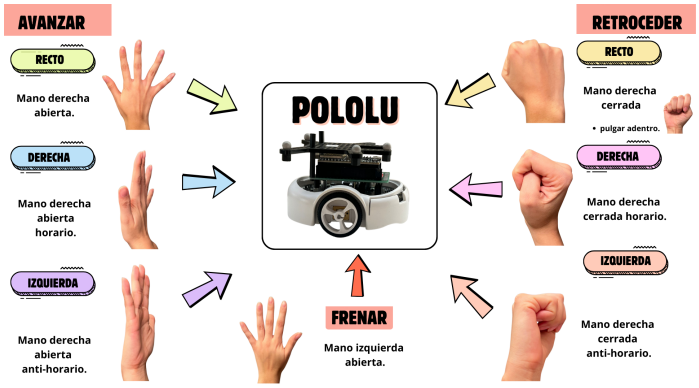
El programa desarrollado en Python implementa un sistema de control en tiempo real para el robot móvil Pololu 3Pi+. En su etapa inicial, el código establece la conexión con el robot, ofreciendo dos alternativas: A través de su dirección *IP* o bien mediante su *ID*. Una vez establecida la comunicación, se definió un esquema de mapeo que asocia cada gesto de la mano con un comando específico de velocidad para las ruedas, permitiendo que el robot replique los movimientos detectados por el sensor Leap Motion.

Para garantizar la estabilidad del sistema, se implementó un mecanismo de *Keep-alive:* [29], encargado de reenviar periódicamente el último comando ejecutado. Adicionalmente, se incorporó un protocolo de seguridad mediante el cual cualquier excepción detectada en la comunicación provoca que el robot se detenga de manera inmediata. Con ello se asegura que, ante una falla, el sistema no continúe en movimiento de forma indeseada, preservando la seguridad del entorno de prueba.

El programa habilita así un control natural e intuitivo del robot, donde los gestos de la mano reemplazan la necesidad de contacto físico con el dispositivo. Para complementar esta arquitectura, se utilizó la plataforma *PlatformIO*, la cual permitió configurar el microcontrolador *ESP32* del robot para conectarse a una red *Wi-Fi* específica.

El esquema de control se definió como se muestra en la Figura 24

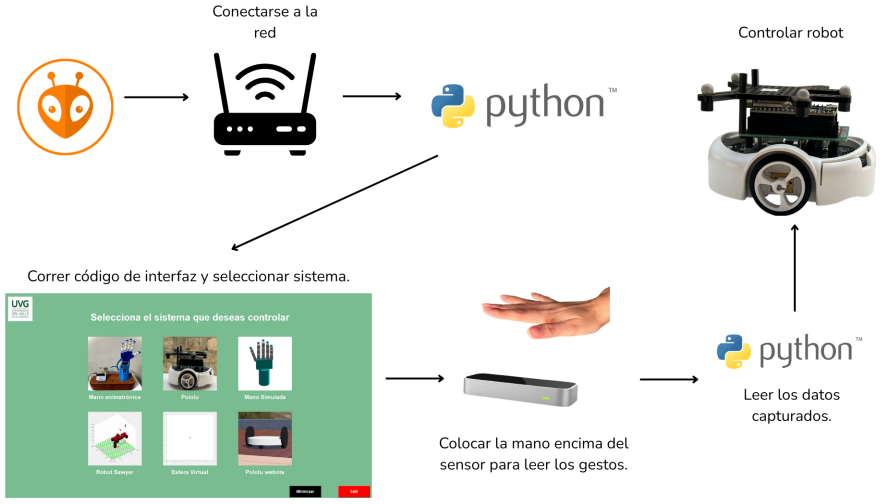
Figura 24. Gestos del robot Pololu 3Pi+ físico.



Nota. Elaboración propia.

Este flujo de comunicación, que integra el procesamiento de gestos en Python, la transmisión por *Wi-Fi* y la interpretación final en el Pololu 3Pi+, se ilustra en la Figura 25.

Figura 25. Diagrama de comunicación e interración del robot Pololu 3Pi+ físico.



Nota. La imagen muestra el diagrama de comunicación, conexión e interacción con Leap Motion del robot Pololu 3Pi+ físico. Elaboración propia

CAPÍTULO 11

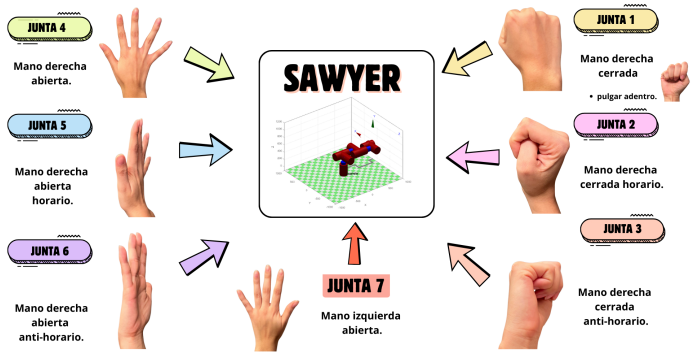
Control del robot Sawyer

11.1. Robot Sawyer en simulación

Se desarrolló una simulación del robot Sawyer en el entorno de Matlab, como se ilustra en la Figura 27. El objetivo de esta implementación fue establecer un control gestual directo, en el cual cada gesto captado por el sensor Leap Motion se traduce en el movimiento de una junta específica del robot.

El sistema de mapeo se definió como se muestra en la Figura 26

Figura 26. Gestos del robot Sawyer simulado.

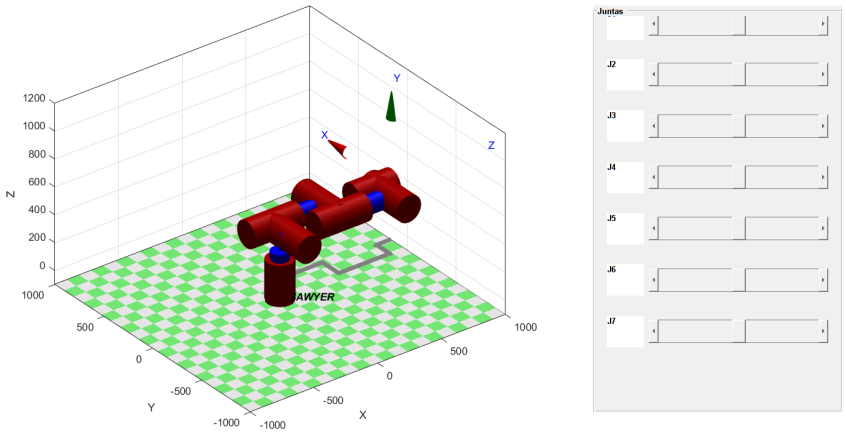


Nota. La imagen muestra los gestos el robot Sawyer simulado. Elaboración propia.

Con esta configuración, se logró que cada articulación del robot pudiera ser manipulada

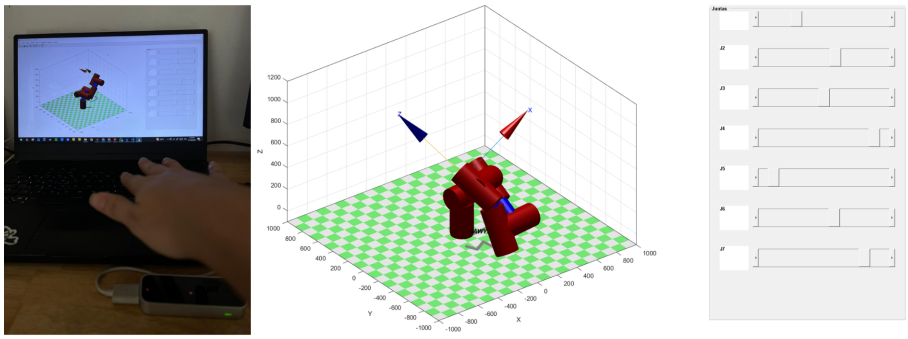
de forma independiente mediante gestos naturales de las manos (Figura 28). Este enfoque permitió validar la viabilidad de utilizar interfaces gestuales para el control de robots colaborativos, ofreciendo una interacción intuitiva y sin contacto físico entre el usuario y el sistema simulado.

Figura 27. Robot Sawyer.



Nota. La imagen muestra el robot Sawyer programado en programa de Matlab junto a un panel de controles deslizantes. Elaboración propia.

Figura 28. Robot Sawyer en distintas posiciones.

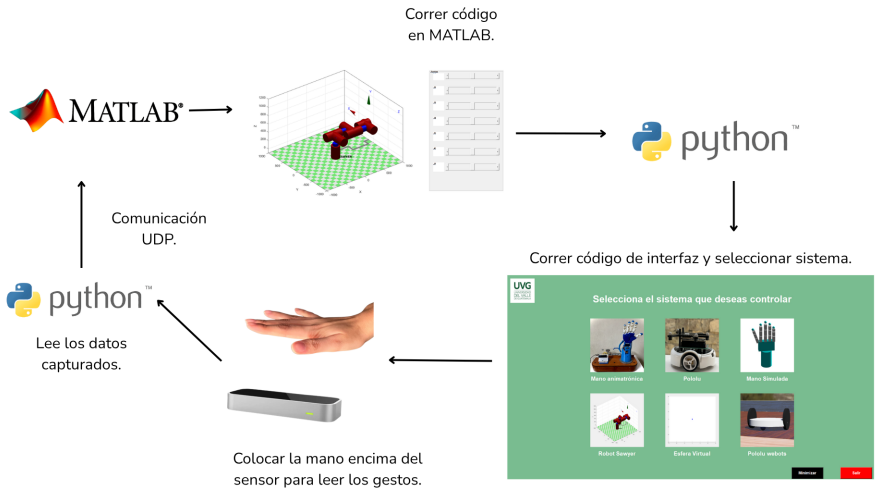


Nota. La imagen muestra el robot Sawyer ubicado en distintas posiciones y controlado por la mano. Elaboración propia.

11.1.1. Control de Robot Sawyer en el programa de Python

En el programa de Python se realizó la captura de los gestos detectados por el sensor Leap Motion, los cuales posteriormente fueron transmitidos hacia el programa de Matlab utilizando el protocolo UDP, de manera análoga al esquema implementado en la simulación de la esfera virtual. Este canal de comunicación permitió enviar en tiempo real los comandos gestuales, asegurando que la información llegara al entorno de simulación del robot Sawyer, como se ilustra en la Figura 29.

Figura 29. Diagrama de comunicación e interración del robot Sawyer físico.



Nota. La imagen muestra el diagrama de comunicación, conexión e interacción con Leap Motion del robot Sawyer simulado en Matlab. Elaboración propia.

11.1.2. Control de Robot Sawyer en el programa de Matlab

En Matlab se implementó un *script* que define la estructura cinemática del robot Sawyer mediante los parámetros de Denavit–Hartenberg, inicializando sus siete articulaciones en posición cero. La interfaz gráfica se diseñó en una ventana que integra la visualización del robot junto con un panel de *sliders*, cada uno correspondiente a una articulación. De este modo, es posible observar cómo los gestos recibidos desde el programa de Python se reflejan en los movimientos de cada junta.

Adicionalmente, se incorporó un mecanismo de rebote que invierte la dirección del movimiento al alcanzar los límites articulares predefinidos. Este recurso asegura que la simulación respete las restricciones físicas del robot real, evitando desplazamientos irreales y manteniendo la consistencia con su comportamiento esperado en un entorno físico.

CAPÍTULO 12

Talleres y guía para actividades STEM

Como parte del desarrollo y difusión del proyecto, se llevaron a cabo una serie actividades para futuros talleres que incluyeron diversas actividades prácticas orientadas a la interacción con los diferentes sistemas robóticos implementados. El objetivo de estas actividades será permitir que los participantes experimentaran de forma directa el control de los robots mediante gestos.

Para facilitar la replicabilidad de los talleres, se elaboró una guía técnica detallada dirigida a los responsables de su ejecución. Esta guía incluyó instrucciones paso a paso sobre cómo conectar el proyecto a diferentes computadoras, desde la instalación de dependencias de software hasta la configuración de redes y la ejecución de los *scripts* de control. Asimismo, se incorporaron recomendaciones sobre la verificación de la comunicación con el sensor, la selección adecuada de puertos de conexión y la solución de problemas comunes que podrían surgir durante la práctica.

Las actividades de los talleres se diseñaron con un enfoque didáctico y progresivo. En una primera etapa, los participantes deben de interactuar con simulaciones sencillas, como el agente puntual, con el fin de familiarizarse con la interfaz de gestos y comprender la lógica de control. Posteriormente, se avanza hacia sistemas más complejos, como la mano animatrónica o el *Pololu*, tanto en su versión simulada como en la física, con el propósito de demostrar las aplicaciones prácticas de la tecnología desarrollada.

12.1. Propuesta de actividades para taller

El taller desarrollado incluyó diversas actividades diseñadas para cada uno de los sistemas robóticos, como se muestra en el Anexo 16.3.

Con el fin de estandarizar la dinámica del taller y apoyar al instructor durante su ejecución, se elaboró una guía para la facilitar las actividades con el sensor Leap Motion y los robots del sistema. Esta guía se encuentra disponible en el Anexo 16.4.

En el caso particular del robot Pololu, se definieron cuatro actividades principales, ya que, por la naturaleza del sistema, ofrece un mayor rango de opciones de interacción. Las actividades propuestas fueron las siguientes:

- Competencia de carreras.
- Evasión de obstáculos.
- Estacionamiento en retroceso.
- Trazado de una figura o forma específica.

Estas dinámicas buscan motivar a personas externas a la universidad a involucrarse en este tipo de proyectos, despertando su interés en estudiar en la Universidad del Valle de Guatemala, especialmente la carrera de Ingeniería Mecatrónica.

Se prevé la implementación de estos talleres dentro de actividades STEM organizadas por la institución, tales como Experiencia UVG, Curso de Vacaciones Mujeres en Ingeniería y Open House. Estos espacios no solo permiten dar a conocer la universidad y sus programas académicos, sino que también promueven la inclusión y la diversidad. En particular, el *Curso de Vacaciones Mujeres en Ingeniería* busca impulsar la participación femenina en áreas tecnológicas, contribuyendo al cierre de la brecha de género en el campo de la mecatrónica.

12.2. Guía técnica de conexiones y solución de problemas

Se elaboró una guía técnica con el propósito de facilitar la interacción de los usuarios con los programas y sistemas robóticos desarrollados en el proyecto como se muestra en el Anexo 16.4. El documento incluye un apartado sobre los posibles problemas que pueden surgir durante el proceso de conexión, junto con sus respectivas soluciones, de manera que quienes dirigen la actividad cuenten con un recurso de apoyo al enfrentar incidencias técnicas.

Asimismo, la guía presenta un listado con los nombres de los *scripts* correspondientes a cada robot y la plataforma en la que se ejecutan, lo que permite identificar rápidamente el archivo correcto para iniciar cada simulación o control físico. De igual forma, se especifican los gestos que deben emplearse en cada sistema robótico, garantizando que los usuarios comprendan de cómo interactuar con cada uno de ellos.

En conjunto, este material busca servir como una referencia práctica y accesible, que reduce la complejidad de las pruebas y optimiza la experiencia del usuario en la utilización de los distintos sistemas robóticos que conforman el proyecto.

CAPÍTULO 13

Conclusiones

A partir del desarrollo del presente trabajo de graduación, se establecen las siguientes conclusiones:

- Se cumplió el objetivo general de diseñar e implementar un sistema de control de dispositivos robóticos mediante gestos utilizando el sensor Leap Motion, integrando plataformas simuladas y físicas de manera funcional.
- Los objetivos específicos fueron alcanzados: Se investigaron las características del Leap Motion, se desarrollaron métodos de extracción y análisis de datos de gestos, se tradujo la información proporcionada por el sensor a comandos para el control de dispositivos robóticos disponibles en la UVG, se implementaron canales de comunicación entre plataformas (UDP, serial y archivos temporales), se diseñó una interfaz gráfica en Python, y se elaboraron talleres para su aplicación en actividades STEM.
- Se demostró que el uso de Leap Motion permite establecer un control gestual efectivo sobre diferentes sistemas robóticos.
- Se identificó que los entornos simulados (mano virtual, esfera en Matlab, Pololu en Webots y Sawyer en Matlab) constituyen una etapa esencial para validar los algoritmos de control antes de su implementación en sistemas físicos, reduciendo riesgos y tiempo de depuración.
- En el caso de los sistemas físicos, se comprobó la viabilidad del control gestual sobre la mano animatrónica y el robot Pololu, aunque se presentaron limitaciones: En la mano animatrónica, fallas mecánicas en el sistema de transmisión; y,en el Pololu físico, retrasos de comunicación que obligaron a simplificar la arquitectura.
- La interfaz gráfica centralizada en Python permitió consolidar en un mismo entorno los diferentes sistemas, facilitando la interacción de los usuarios y contribuyendo a la usabilidad del sistema.

- El desarrollo de talleres y de una guía técnica evidenció el potencial del sistema como herramienta educativa y de divulgación científica, aportando una solución innovadora para actividades STEM en la Universidad del Valle de Guatemala.

CAPÍTULO 14

Recomendaciones

A partir de los resultados y las limitaciones identificadas, se sugiere lo siguiente:

- Mejorar el sistema mecánico de la mano animatrónica, ya que los motores realizan el movimiento, pero los dedos no responden de manera adecuada. Esto podría estar relacionado con el mecanismo de transmisión por hilos, el cual requiere un rediseño para garantizar una mayor eficiencia en la transferencia del movimiento.
- Integrar una pinza al robot Pololu que pueda ser controlada mediante la otra mano. De esta manera, el robot no solo sería capaz de desplazarse hacia determinados objetos, sino también manipularlos, ampliando sus aplicaciones.
- Explorar el desarrollo de robótica de enjambre, en la que múltiples robots Pololu respondan de manera coordinada a los gestos detectados. Cada gesto podría asociarse con una acción específica, lo que permitiría experimentar con dinámicas colaborativas entre varios agentes robóticos.
- Implementar un sistema basado en visores de realidad virtual como Oculus, que permita controlar distintos objetos mediante el movimiento de la cabeza en conjunto con el sensor Leap Motion [30]. Esta integración abriría posibilidades en entornos inmersivos e interactivos.
- Investigar la posibilidad de controlar drones utilizando el sensor Leap Motion. Esto permitiría validar el desempeño del sistema en plataformas aéreas, expandiendo su aplicación más allá de robots móviles terrestres.

Estas recomendaciones buscan fortalecer la línea de investigación en interfaces naturales para el control de sistemas robóticos y abrir nuevas oportunidades de aplicación en áreas como manipulación, coordinación multi-robot y entornos inmersivos.

CAPÍTULO 15

Referencias

[1] Y. Du, S. Liu, L. Feng, M. Chen y J. Wu, «Hand Gesture Recognition with Leap Motion,» nov. de 2017. dirección: http://arxiv.org/abs/1711.04293.

[2] R. McCartney, J. Yuan y H.-P. Bischof, *Gesture Recognition with the Leap Motion Controller*, 2015. dirección: https://www.leapmotion.com/.

[3] M. Rajesh, D. Savatekar y M. A. A. Dum, *Design Of Control System For Articulated Robot Using Leap Motion Sensor*, mar. de 2016. dirección: www.irjet.net.

[4] M. F. Girón, «Interfaz Biomédica para el Control de Sistemas Robóticos Utilizando Señales EMG,» Trabajo de graduación de licenciatura, Universidad del Valle de Guatemala, 2020.

[5] S. J. Rivera, «Diseño e implementación de interfaz biomédica y sensores inerciales para el control de actuadores en tiempo real,» Trabajo de graduación de licenciatura, Universidad Del Valle de Guatemala, 2025.

[6] S. Andrade, «Interfaz de control de sistemas robóticos mediante sensor Leap Motion para fines educativos,» Proyecto de graduación, Tesis de licenciatura, Universidad del Valle de Guatemala, 2024.

[7] C. A. Acosta, R. E. Villacrés, J. C. Calderón y M. J. García, «A Survey on Human–Machine Interaction in Industry 4.0,» *Sensors*, vol. 19, n.º 1072, págs. 1-20, 2019. DOI: 10.3390/s19051072. dirección: https://www.mdpi.com/1424-8220/19/5/1072.

[8] F. Weichert, D. Bachmann, B. Rudak y D. Fisseler, «Analysis of the Accuracy and Robustness of the Leap Motion Controller,» *Sensors*, vol. 13, n.º 5, págs. 6380-6393, 2013. DOI: 10.3390/s130506380. dirección: https://www.mdpi.com/1424-8220/13/5/6380.

[9] Ultraleap. «LeapC API Concepts – Ultraleap Tracking API. »dirección: https://docs.ultraleap.com/api-reference/tracking-api/leapc-guide/leap-concepts.html.

[10] P. M. Oliveira, «Analysis and Evaluation of Gesture Recognition using LeapMotion,» en *Proceedings of the 10th Doctoral Symposium in Informatics Engineering (DSIE'15)*, Faculty of Engineering, University of Porto, 2015, págs. 83-92, ISBN: 978-972-752-173-9.

[11] Ultraleap. «Leap Motion Controller - Downloads and Documentation. »dirección: https://www.ultraleap.com/downloads/leap-controller/.

[12] Ultraleap. «Leap Motion Controller Performance Modes (Hyperion). »dirección: https://docs.ultraleap.com/hand-tracking/Hyperion/performancemodes.html.

[13] X. Zhang, R. Zhang, L. Chen y X. Zhang, «Natural Gesture Control of a Delta Robot Using Leap Motion,» *Journal of Physics: Conference Series*, vol. 1187, 2019. DOI: 10.1088/1742-6596/1187/3/032042.

[14] F. M. Hernández, «El Concepto de Distancia y su Aplicación en Estadística Multivariada,» *Boletín*, n.º 27, 2015. dirección: https://www.researchgate.net/publication/267820077.

[15] OBS Business School. «Datagrama, características a tener en cuenta.» Accedido el 24 de septiembre de 2025. dirección: https://www.obsbusiness.school/blog/datagrama-caracteristicas-tener-en-cuenta.

[16] W. R. Stevens y K. R. Fall, *TCP/IP Illustrated, Volume 1: The Protocols*, 2nd. Addison-Wesley, 2012, ISBN: 978-0-321-33631-6.

[17] J. Postel, *User Datagram Protocol*, RFC 768, 1980. dirección: https://www.rfc-editor.org/rfc/rfc768.html.

[18] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*, 10th. John Wiley & Sons, 2018, ISBN: 978-1-119-32091-3.

[19] R. M. Murray, Z. Li y S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994, ISBN: 978-0849379819.

[20] F. J. Somma, «Cuaterniones y ángulos de Euler para describir rotaciones en $\mathbb{R}^3$,» Directora: Dra. María Lorena Bergamini, Tesis de Licenciatura en Matemática, Universidad Abierta Interamericana, Facultad de Tecnología Informática, Buenos Aires, Argentina, 2018.

[21] Ultraleap. «Getting Started with Ultraleap Hand Tracking: Device Orientation.» Accedido el 24 de septiembre de 2025. dirección: https://docs.ultraleap.com/hand-tracking/getting-started.html#device-orientation.

[22] K. Arulkumaran, M. P. Deisenroth, M. Brundage y A. A. Bharath, «A Brief Survey of Deep Reinforcement Learning,» *arXiv preprint arXiv:1708.05866*, 2017. dirección: https://arxiv.org/abs/1708.05866.

[23] OpenAI, *ChatGPT (Sep 27 version) [Large language model]*, https://chat.openai.com/, 2025.

[24] Ó. Gálvez, «Optimización de diseño de una mano y muñeca animatrónica antropomórfica de la fase tres e implementación de un control interactivo,» Disponible en Repositorio Institucional UVG, Tesis de Licenciatura, Universidad del Valle de Guatemala, 2021. dirección: https://repositorio.uvg.edu.gt/handle/123456789/3879.

[25] M. García, «Optimización del proyecto de una mano y muñeca animatrónica antropomórfica y su sistema de control y operación,» Disponible en Repositorio Institucional UVG, Tesis de Licenciatura, Universidad del Valle de Guatemala, 2022. dirección: https://repositorio.uvg.edu.gt/handle/123456789/4242.

[26] L. Wang, T. Meydan y P. Williams, «A Two-Axis Goniometric Sensor for Tracking Finger Motion,» *Sensors*, vol. 17, abr. de 2017. DOI: 10.3390/s17040770.

[27] P. A. Kirkpatrick. «3.3. Pipes and FIFOs — Computer Systems Fundamentals.» Accedido el 24 de septiembre de 2025. dirección: https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/Pipes.html.

[28] M. L. Laboratory, *Large Scale Parallelization Using File-Based Communications*.

[29] Arsys. «¿Qué es el servicio Keep-Alive y cómo ayuda a tu web?» Accedido el 24 de septiembre de 2025. dirección: https://www.arsys.es/blog/que-es-el-servicio-keep-alive-y-como-ayuda-a-tu-web.

[30] Ultraleap. «XR Setup for Leap Motion Controller. »dirección: https://docs.ultraleap.com/hand-tracking/XR-Setup/XR-setup.html#for-leap-motion-controller.

[31] Lebalap Academy. «Circuito de Mónaco (GP de Mónaco F1). »dirección: https://lebalap.academy/f1/circuito-de-monaco/.

[32] Lebalap Academy. «Circuito de Suzuka (GP de Japón F1). »dirección: https://lebalap.academy/f1/circuito-de-suzuka/.

CAPÍTULO 16

Anexos

16.1. Paquetes instalados en el entorno virtual

Cuadro 1. Paquetes instalados en el entorno de desarrollo

Paquete	**Versión**
build	1.2.2.post1
cffi	1.17.1
colorama	0.4.6
future	1.0.0
importlib_metadata	8.5.0
iso8601	2.1.0
numpy	1.24.4
opencv-python	4.11.0.86
pillow	10.4.0
pip	25.0.1
pycparser	2.22
pyproject_hooks	1.2.0
pyserial	3.5
pywin32	311
PyYAML	6.0.2
setuptools	75.3.2
tomli	2.2.1
wheel	0.45.1
zipp	3.20.2

Nota. Elaboración propia.

16.2. Módulos de interfaz gráfica

Figura 30. Interfaz de mano animatrónica.

Nota. La imagen muestra el modulo de instrucciones de la interfaz de la mano animatrónica. Elaboración propia.

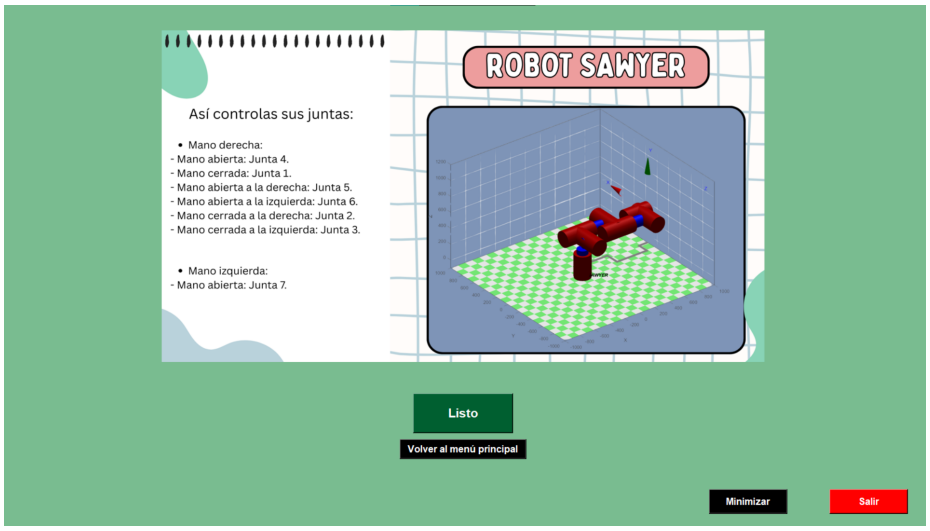
Figura 31. Interfaz Pololu 3Pi+ físico.

Nota. La imagen muestra el modulo de instrucciones de la interfaz del Pololu 3Pi+ físico. Elaboración propia.

Figura 32. Interfaz de mano simulada.

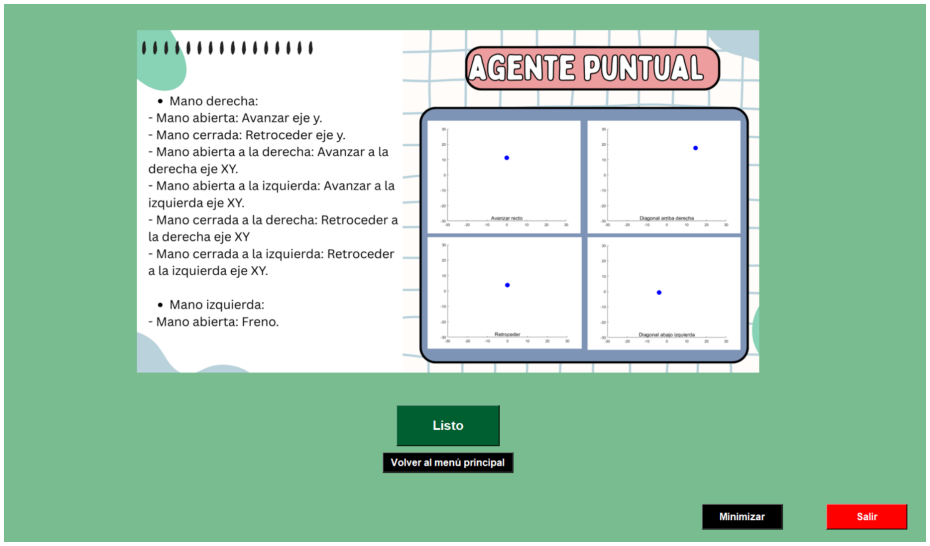
Nota. La imagen muestra el modulo de instrucciones de la interfaz de la mano simulada. Elaboración propia.

Figura 33. Interfaz del robot simulado Sawyer.



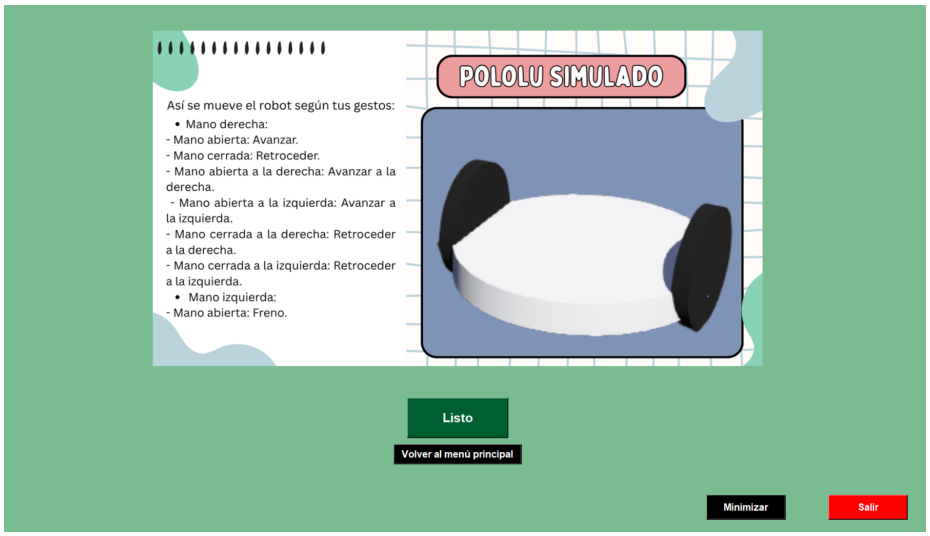
Nota. La imagen muestra el modulo de instrucciones de la interfaz del robot Sawyer. Elaboración propia.

Figura 34. Interfaz del agente puntual.



Nota. La imagen muestra el modulo de instrucciones de la interfaz del agente puntual. Elaboración propia.

Figura 35. Interfaz Pololu 3Pi+ simulado.



Nota. La imagen muestra el modulo de instrucciones de la interfaz del Pololu 3Pi+ simulado. Elaboración propia.

16.3. Talleres

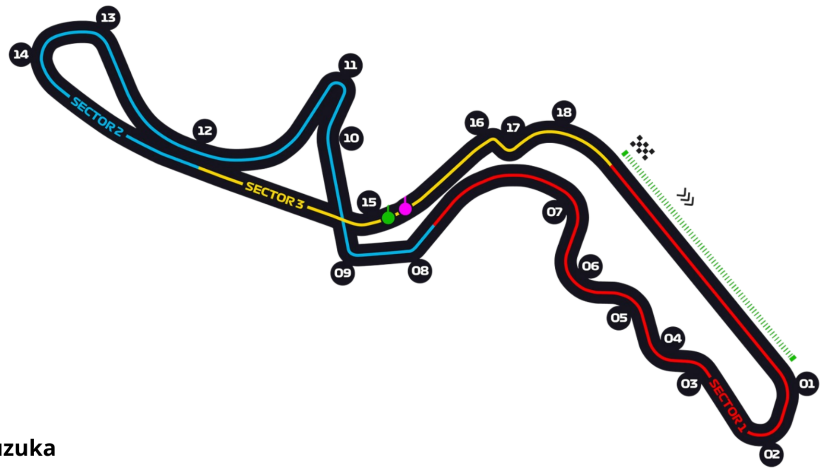
Figura 36. Taller Pololu.

Nota. La imagen muestra el *índice* del Taller del Pololu físico. Elaboración propia.

Figura 37. Taller Pololu, carreras.

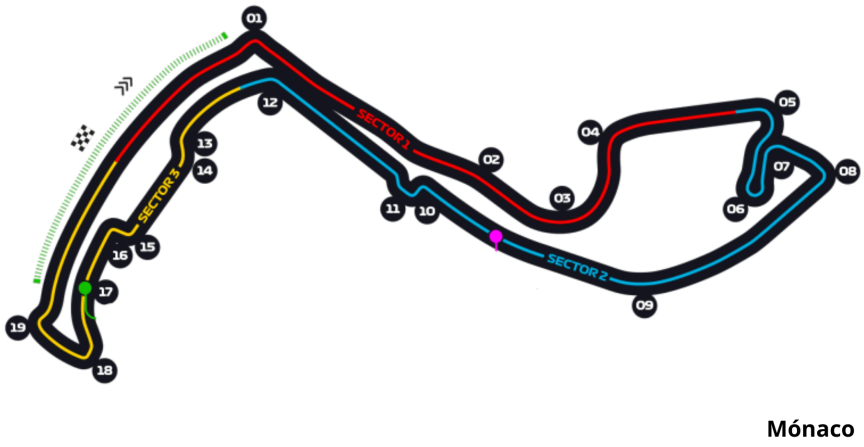
Nota. La imagen muestra las *instrucciones* del Taller del Pololu físico para la actividad de carreras. Elaboración propia.

Figura 38. Taller Pololu, pista Suzuka.



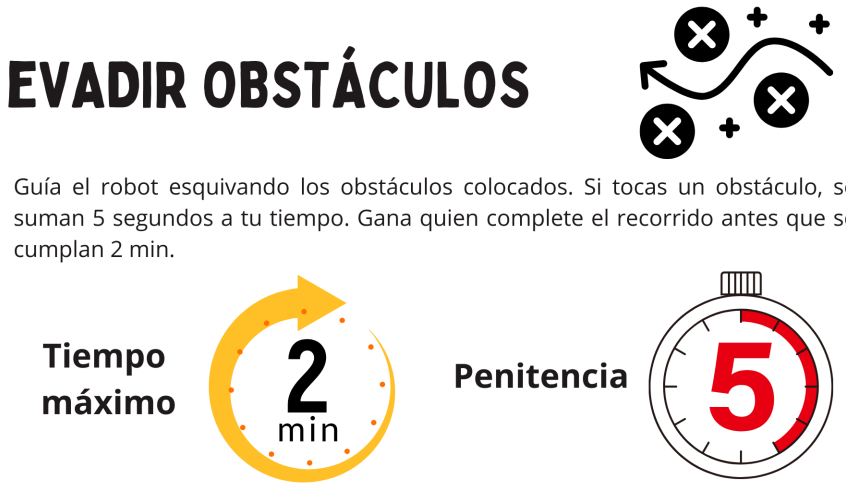
Nota. La imagen muestra la *pista Suzuka* del Taller del Pololu físico para la actividad de carreras.Esta imagen fue obtenida de [31].

Figura 39. Taller Pololu, pista Mónaco.



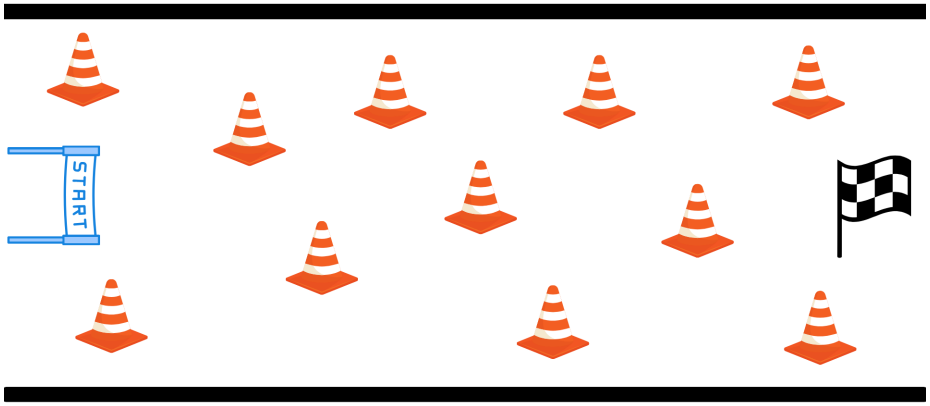
Nota. La imagen muestra las *pista Mónaco* del Taller del Pololu físico para la actividad de carreras.Esta imagen fue obtenida de [32].

Figura 40. Taller Pololu, evasión de obstáculos.



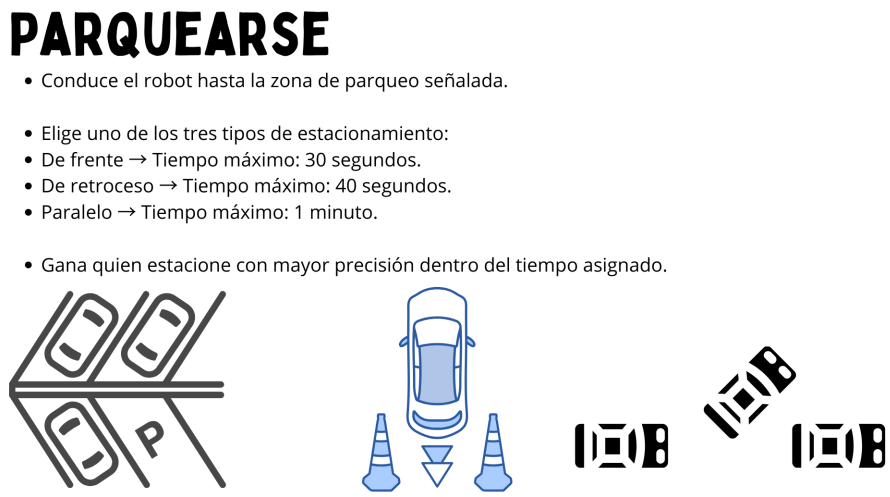
Nota. La imagen muestra las *instrucciones* del Taller del Pololu físico para la actividad de evadir obstáculos. Elaboración propia.

Figura 41. Taller Pololu, obstáculos.



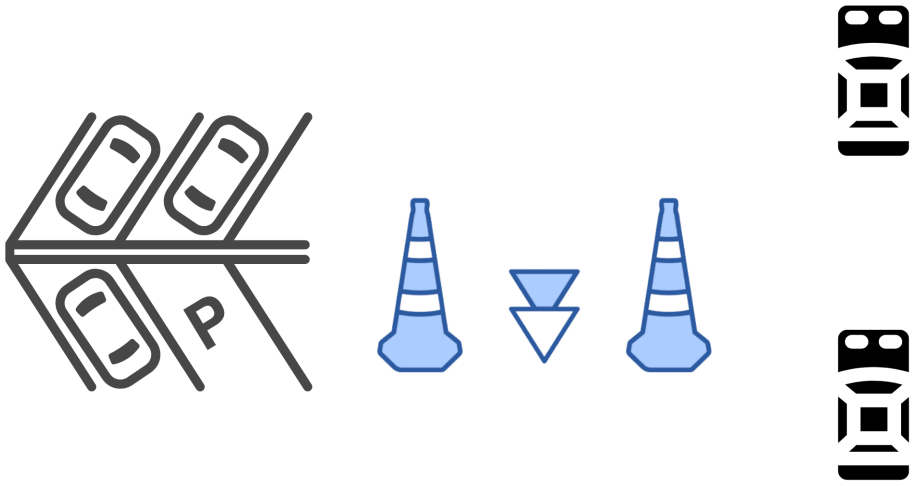
Nota. La imagen muestra los *obstáculos* a evadir del Taller del Pololu físico para la actividad de evadir obstáculos. Elaboración propia.

Figura 42. Taller Pololu, parquearse.



Nota. La imagen muestra las *instrucciones* del Taller del Pololu físico para la actividad de parquearse de retroceso, paralelo y de frente. Elaboración propia.

Figura 43. Taller Pololu, parqueos.



Nota. La imagen muestra los distintos *parqueos* del Taller del Pololu físico para la actividad de parquearse de retroceso, paralelo y de frente. Elaboración propia.

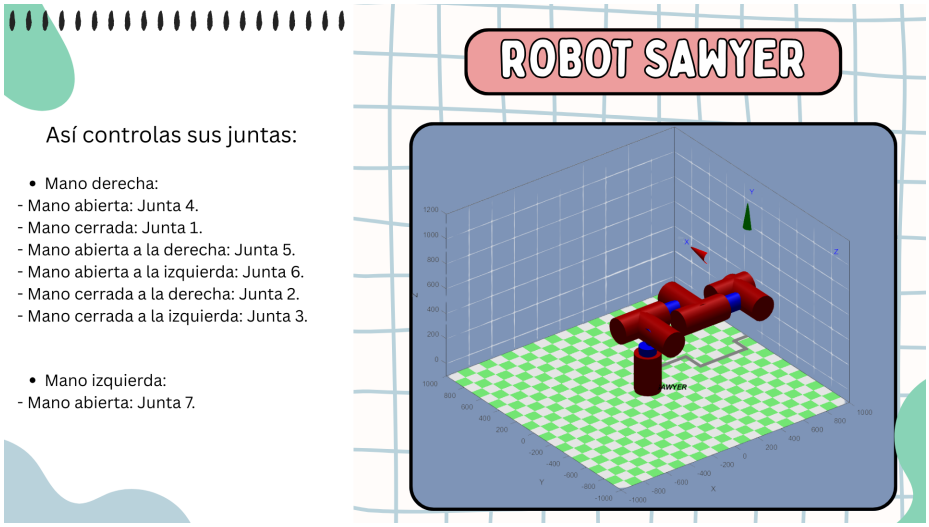
Figura 44. Taller Pololu, trazo de una forma.

Nota. La imagen muestra las *instrucciones* del Taller del Pololu físico para la actividad de trazar formas. Elaboración propia.

Figura 45. Taller Pololu, formas a trazar.

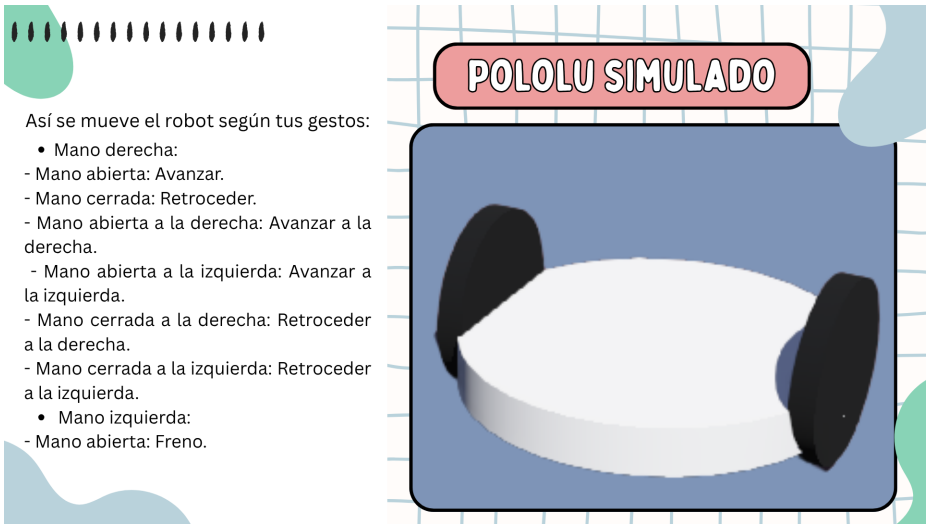
Nota. La imagen muestra las *formas* a trazar del Taller del Pololu físico para la actividad de trazar formas. Elaboración propia.

Figura 46. Taller del robot Sawyer simulado.



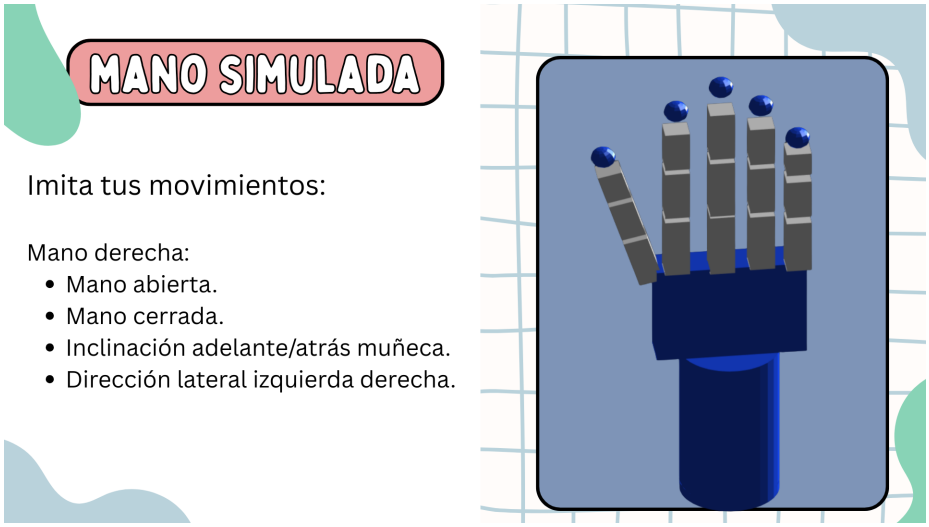
Nota. La imagen muestra el taller del robot *sawyer* simulado que consta de sus instrucciones para utilizarlo. Elaboración propia.

Figura 47. Taller Pololu, Webots.



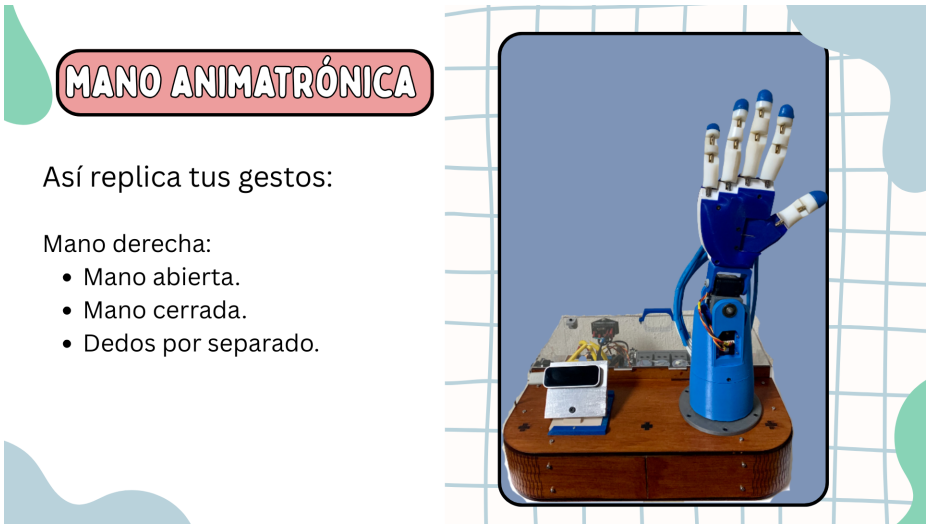
Nota. La imagen muestra el taller del Pololu en *Webots* que consta de sus instrucciones para utilizarlo. Elaboración propia.

Figura 48. Taller de mano simulada.



Nota. Elaboración propia.

Figura 49. Taller de mano animatrónica.



Nota. Elaboración propia.

16.4. Guía para taller con sensor Leap Motion y guía de conexiones

Este anexo reúne las guías desarrolladas para el uso completo del sistema robótico basado en el sensor Leap Motion. Incluye la guía para taller con el sensor Leap Motion, orientada a actividades educativas y demostraciones, así como la guía técnica de conexiones, que describe

los procedimientos de instalación, configuración y operación del sistema.

Los documentos completos pueden consultarse en el siguiente enlace:

- Repositorio en GitHub de PDFs de guías

16.5. Github

En este anexo se incluye el repositorio en línea donde se encuentra el código fuente y materiales adicionales del proyecto:

- `https://github.com/MarcelaPadillaMicheo/Proyecto-de-graduaci-n.git`

CAPÍTULO 17

Glosario

Datagrama: Unidad de datos asociada con la conmutación de paquetes. Su uso permite un servicio de comunicación sin conexión a través de una red de conmutación de paquetes. 14

Keep-alive: Funcionalidad que permite mantener una conexión persistente entre el servidor y el cliente. 37

Pipes: Mecanismo de comunicación entre procesos que permite el envío de un flujo de bytes unidireccional a través de una tubería. 33