Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Diseño de un controlador para un brazo robótico mediante detección de movimiento

INGE-2605

Proyecto Integrador

Previo la obtención del Título de:

Nombre de la titulación

Ingeniero en Mecatrónica

Presentado por:

Luis Eduardo Zambrano Ortiz

Walter Azael Sánchez Merchán

Guayaquil - Ecuador

Año: 2024

Declaración Expresa

Nosotros Luis Eduardo Zambrano Ortiz y Walter Azael Sánchez Merchán acordamos y reconocemos que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de mi/nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 22 de mayo del 2024.

Walter Azael Sánchez Merchán

Luis Eduardo Zambrano Ortiz

Evaluadores

Firmado electrónicamente por:
JONATHAN ROBERTO
LEON TORRES

Jonathan Roberto Leon Torres, M. Sc.

Profesor de Materia

Firmado electrónicamente por:
JORGE LUIS HUREL
EZETA

Jorge Luis Hurel Ezeta, PhD

Tutor de proyecto

Resumen

Este proyecto busca desarrollar un controlador de posición para un brazo robótico usando una cámara Kinect v1. El objetivo es lograr un control preciso y en tiempo real mediante gestos naturales. Se plantea como hipótesis que un controlador PI, en combinación con la librería SimpleOpenNI para procesamiento de datos, permite un control intuitivo y efectivo del brazo. La justificación radica en la accesibilidad y facilidad de uso que proporciona esta metodología. En el desarrollo, se utilizó una cámara Kinect v1 para capturar el movimiento del usuario, obteniendo coordenadas tridimensionales de las articulaciones. Se empleó un controlador PI para ajustar la velocidad y posición del brazo robótico, minimizando el error entre la posición actual y la deseada. Los materiales incluyen un brazo robótico, la cámara Kinect y la librería SimpleOpenNI para el procesamiento de datos. Los resultados muestran un control preciso del brazo robótico mediante gestos, confirmando la hipótesis planteada. Se concluye que el sistema permite un control eficiente y en tiempo real.

Palabras clave: control de posición, Kinect, mapeado de articulaciones, parámetros PI.

Abstract

This project aims to develop a position controller for a robotic arm using a Kinect v1 camera. The objective is to achieve precise and real-time control through natural gestures. The hypothesis is that a PI controller, in combination with the SimpleOpenNI library for data processing, enables intuitive and effective control of the arm. The justification lies in the accessibility and ease of use provided by this methodology. During development, a Kinect v1 camera was used to capture the user's movement, obtaining three-dimensional coordinates of the joints. A PI controller was employed to adjust the speed and position of the robotic arm, minimizing the error between the current and desired positions. The materials include a robotic arm, the Kinect camera, and the SimpleOpenNI library for data processing. The results demonstrate precise control of the robotic arm through gestures, confirming the hypothesis. It is concluded that the system allows for efficient and real-time control.

Keywords: position control, Kinect, joint mapping, PI parameters.

Índice general

Abreviaturas

ESPOL Escuela Superior Politécnica del Litoral

GDL Grados de libertad

CNC Control numérico por computadora

SCARA Selective Compliance Assembly Robot Arm

EOAT End of Arm Tooling

PS Playstation

IDE Integrated Development Environment

MATLAB Matrix Laboratory

ARM Advanced RISC Machine

PIC Peripheral Interface Controller

AVR Advanced Virtual RISC

CPU Central Processing Unit

3D Three-Dimensional

AI Artificial Intelligence

PID Proporcional, Integral, Derivativo

Índice de figuras

Índice de tablas

Capítulo 1

1.1 Introducción

La intersección entre la tecnología y la ingeniería ha desencadenado una revolución en el campo de la robótica, donde los avances en la percepción y el control han permitido la creación de sistemas cada vez más sofisticados y adaptativos [1], [2]. En este contexto, el control de brazos robóticos mediante cámaras de movimiento se erige como un área de investigación fascinante y prometedora. Esta tesis se enmarca en este emocionante dominio, donde se aborda el desafío de fusionar la capacidad perceptiva de las cámaras con la precisión del control robótico para lograr una interacción intuitiva y eficiente entre humanos y máquinas.

El control de un brazo robótico a través de una cámara de movimiento representa un paradigma innovador que busca emular la destreza humana en la manipulación de objetos. Inspirado por la notable habilidad de los seres humanos para interpretar y responder a las señales visuales, este enfoque busca aprovechar la información visual capturada por la cámara para guiar los movimientos del brazo robótico de manera precisa y adaptable. El desafío es desarrollar algoritmos y técnicas de control que permitan traducir eficientemente la información visual en comandos ejecutables para el brazo robótico, garantizando una interacción fluida y segura en entornos dinámicos y complejos [3], [4].

El contexto de esta investigación se sitúa en un escenario donde la colaboración estrecha entre humanos y robots es esencial, ya sea en entornos industriales, médicos o domésticos. En estos contextos, controlar un brazo robótico de manera intuitiva y natural mediante una interfaz visual mejora la eficiencia operativa y amplía las posibilidades de aplicación de la robótica en varios campos. Sin embargo, para materializar este potencial, es crucial superar una serie de desafíos técnicos y conceptuales que abarcan desde la percepción visual hasta la planificación y ejecución de movimientos robóticos [5], [6], [7].

En este contexto, esta tesis se propone desarrollar una solución innovadora para el control de brazos robóticos mediante cámaras de movimiento, abordando tanto aspectos teóricos como prácticos. En este campo, con un enfoque multidisciplinario que combina la robótica, la visión por computadora y el control automático, se buscará avanzar en el estado del arte, para contribuir al desarrollo de sistemas robóticos más inteligentes, adaptables y colaborativos [8], [9].

1.2 Descripción del Problema

En la actualidad, la automatización industrial demanda la implementación de sistemas robóticos con alta destreza, capaces de realizar tareas complejas y precisas. Sin embargo, uno de los desafíos más significativos que enfrentan las empresas al integrar brazos robóticos en sus procesos es el largo tiempo de configuración y los elevados costos asociados tanto a la capacitación como a la puesta en marcha de estos sistemas. Operar un brazo robótico con un alto grado de destreza requiere un profundo conocimiento técnico, no solo en la programación y control del robot, sino también en la comprensión de su cinemática y dinámica. Esto implica que los operadores deben pasar por extensos programas de formación para poder manejar eficazmente el robot y optimizar su rendimiento. Además, la necesidad de ajustar y recalibrar los sistemas robóticos en respuesta a cambios en las tareas o en el entorno de trabajo añade otra capa de complejidad, prolongando los tiempos de inactividad y aumentando los costos operativos.

Estos factores limitan la accesibilidad de la robótica avanzada a un espectro más amplio de aplicaciones, especialmente en entornos donde se requiere adaptabilidad y precisión. Por lo tanto, es crucial desarrollar soluciones que no solo reduzcan el tiempo y el costo de capacitación, sino que también simplifiquen la operación del brazo robótico, permitiendo un uso más eficiente y flexible de estos sistemas en diversas industrias.

1.3 Justificación del Problema

El desarrollo de un controlador para brazos robóticos que utilice una cámara de movimiento en tiempo real presenta una solución innovadora para mitigar los desafíos asociados con los largos tiempos y elevados costos de operación y capacitación. Integrar una cámara de movimiento en tiempo real en el sistema de control permitiría una interfaz más intuitiva y accesible para los operadores, simplificando el proceso de aprendizaje y reduciendo la necesidad de una formación técnica exhaustiva.

Una cámara de movimiento en tiempo real podría capturar y seguir los movimientos del operador o de un objeto de referencia, traduciéndolos directamente en comandos para el brazo robótico. Esto permitiría a los operadores interactuar con el robot de manera más natural, como si estuvieran guiando manualmente su movimiento, eliminando la barrera de la programación compleja. Además, este enfoque tendría la capacidad de adaptarse dinámicamente a cambios en el entorno de trabajo, mejorando la eficiencia operativa y reduciendo los tiempos de inactividad.

Al reducir la complejidad de la interfaz de usuario y hacer que el sistema sea más intuitivo, no solo se disminuyen los tiempos de capacitación, sino que también se abre la posibilidad de que un mayor número de operadores, incluso aquellos con menos experiencia técnica, puedan controlar el brazo robótico con alta destreza. Esto no solo optimiza los recursos humanos y económicos, sino que también expande el potencial de aplicación de los brazos robóticos en diversas industrias, permitiendo una adopción más amplia y flexible de la robótica avanzada.

1.4 Objetivos

1.4.1 Objetivo general

Integrar el sistema de movimiento de un brazo robótico con un algoritmo de control que utilice una cámara de movimiento para poder operar el brazo usando gestos.

1.4.2 Objetivos específicos

1. Desarrollar un algoritmo de detección visual capaz de reconocer gestos y movimientos de un usuario para su implementación dentro del sistema del robot.
2. Ensamblar un brazo robótico de seis articulaciones controlado por un microprocesador asegurando su correcto funcionamiento para dotar de movimiento al sistema mecatrónico.
3. Implementar las funciones previstas en una cámara de movimiento capaces de realizar detección de objetos en un entorno físico para integrarlas en un sistema de reconocimiento visual.

1.5 Marco teórico

1.5.1 Robótica a nivel industrial

Existen muchos aspectos que difieren entre el Robot industrial y el manipulador, se origina uno de ellos en el mercado japones que dice que se considera robot industrial es un dispositivo mecánico que cuenta con articulaciones móviles que le permiten el movimiento.

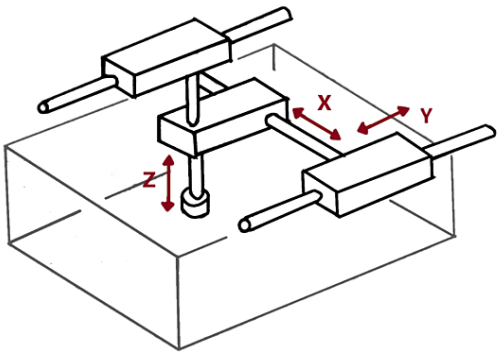
Los tipos de robots industriales que se pueden encontrar son:

Los que se pueden encontrar en el ámbito industrial son el robot cartesiano, robot antropomórfico, robot cilíndrico, robot SCARA, robot Delta, robot esférico. El robot cartesiano está formado por tres articulaciones de forma prismática, tal como se puede observar en la **Figura 1.1,** limitando el movimiento de la herramienta al movimiento lineal basado en los ejes X, Y y Z del sistema de coordenadas tridimensionales [10]. Los robots antropomórficos, que puede ser observado en la **Figura 1.2,** se usan con mayor frecuencia en actividades de fabricación que requieren mayor precisión, como la soldadura, el montaje o el mecanizado [11]. El robot cilíndrico posee un rango de movimiento cilíndrico, tal como lo demuestra la **Figura 1.3**, ya que consta de una junta de revolución y dos prismáticas [13]. El robot SCARA tiene un brazo flexible en el plano horizontal (XY) y rígido en la dirección vertical (Z), los cuales se pueden apreciar en la **Figura**

1.4, con dos juntas de revolución y una prismática, conectados por las juntas de revolución orientadas al mismo eje [14]. El robot Delta consta de al menos tres eslabones conectados a una herramienta de extremo (EOAT) y a una base común, evidenciado en la **Figura 1.5**, con el EOAT unido a los eslabones mediante tres juntas universales no accionadas [15]. Por último, los robots esféricos, utilizan un sistema de coordenadas tridimensionales r, θ y φ, y tienen un alcance esférico, tal como lo demuestra la **Figura 1.6** [16]**.**

Figura 1.1

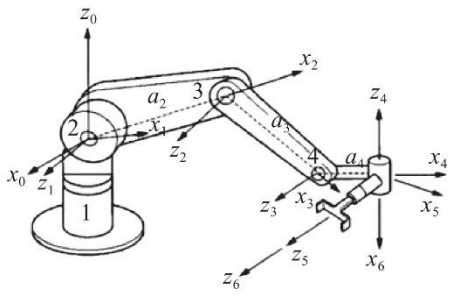
Robot cartesiano



Nota: fuente: Pavía Parra, E. (2016) [10]

Figura 1.2

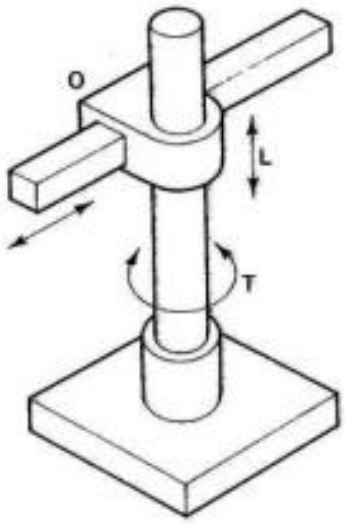
Robot antropomórfico



Nota: Fuente: Huang, L., Dou, Y., Fan, Y., He, Z., Zhang, S., Zhang, L., & Zhang, B. (2022). [11]

Figura 1.3

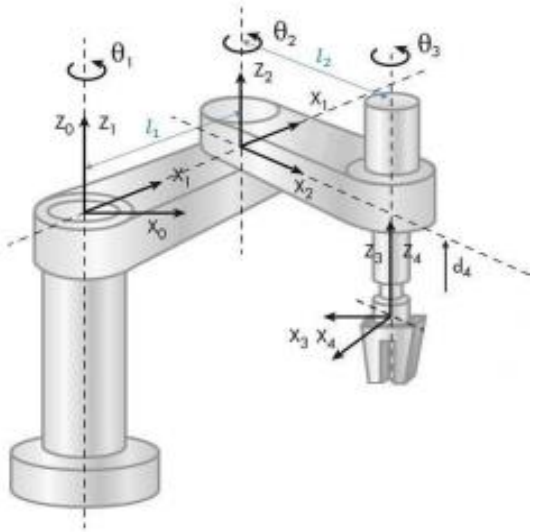
Robot Cilíndrico



Nota: Fuente: Anónimo. Autonomía de los robots. [13]

Figura 1.4

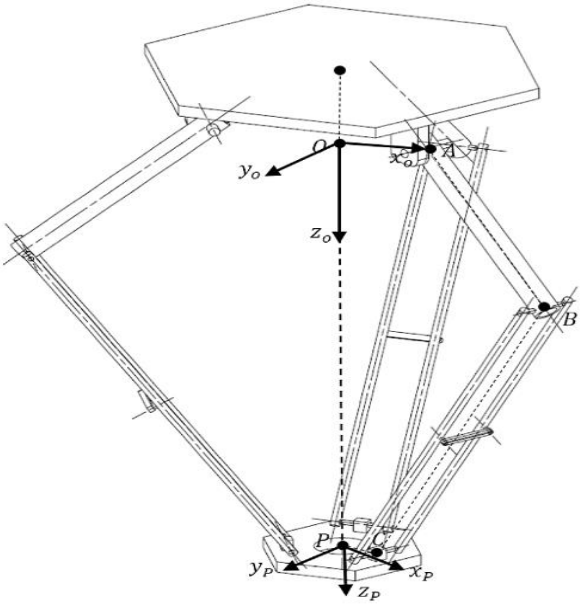
Robot SCARA



Nota: Fuente: ResearchGate. Robot manipulador SCARA. [14]

Figura 1.5

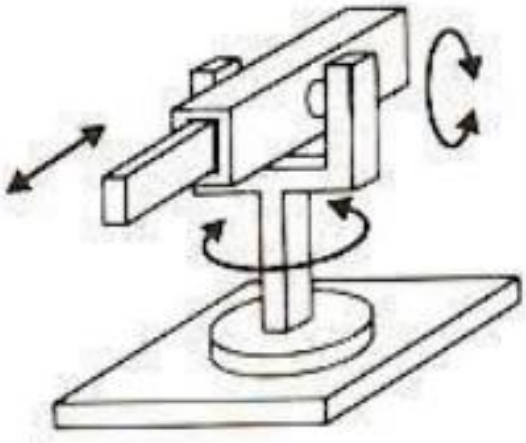
Robot Delta



Nota: Fuente: Londoño, S., Maya, J. P., & Giraldo, J. (2016). [15]

Figura 1.6

Robot Esférico



Nota: Ingeniería de sistemas y automática. Tema 2 Morfología [16]

1.5.2 Aplicaciones de los robots industriales:

Los robots de montaje, inicialmente adoptados en la industria automovilística, son ahora esenciales para el ensamblaje robótico de alta velocidad de piezas pequeñas, ofreciendo un rendimiento y precisión superiores a los del trabajo manual. Los robots dispensadores de adhesivos aplican selladores en diversas tareas, como la fijación de piezas y la aplicación de pegamento y resina epoxi, trabajando de manera compacta y rápida. En la manipulación, estos robots transportan mercancías en almacenes y preparan pedidos para envío, respondiendo a la creciente demanda. Los robots de carga y alimentación de máquinas automatizan el proceso de introducir y retirar piezas de las máquinas, aumentando la eficiencia operativa. En el fresado, los robots cortan y dan forma a materiales, siendo cada vez más automatizados con la llegada de la CNC. Los robots de perforación ofrecen mayor precisión y repetitividad en trabajos manualmente agotadores y peligrosos. Los robots de corte por láser mejoran la precisión y reducen la deformación del material gracias al uso del rayo láser. En soldadura, los robots aumentan la productividad, repetibilidad y precisión en aplicaciones como fontanería, electrónica y joyería. Los robots de fundición a presión automatizan la fabricación de piezas, incluyendo su enfriamiento y limpieza. Finalmente, los robots de pintura y revestimiento mejoran la seguridad al reducir la exposición a humos nocivos, proporcionando resultados uniformes y de alta calidad con una operación continua y sin pausas [17].

Figura 1.7

Robot de agarre automático de una línea

Nota: fuente: VM Systems. (2021, April 14). Robot de picking automático para final de línea. VM Systems. [18]

1.5.3 Cámaras de movimiento:

También conocidas como cámaras de captura o seguimiento de movimiento, estas tecnologías registran en tiempo real el desplazamiento de objetos o personas, utilizando una combinación avanzada de sensores infrarrojos, marcadores ópticos y algoritmos de visión por computadora para captar la posición y el movimiento en tres dimensiones [19]. Diversas empresas reconocidas han lanzado al mercado distintas versiones de estas cámaras. Entre ellas, destacan el sensor Kinect de Microsoft, el PlayStation Move de Sony y el Wii Remote de Nintendo, pionero en esta tecnología.

El sensor Kinect para Xbox 360, desarrollado por Microsoft y lanzado en noviembre de 2010, marcó un hito en la interacción con videojuegos y aplicaciones al permitir el uso de movimientos y comandos de voz, eliminando la necesidad de controladores físicos. Equipado con una cámara RGB, un sensor de profundidad y un conjunto de micrófonos, el Kinect ofrecía reconocimiento facial e interacción en 3D. Aunque su objetivo inicial era mejorar la experiencia de los videojuegos, el Kinect fue rápidamente adoptado en campos como la educación, la salud y proyectos de desarrollo independiente, ejerciendo una influencia significativa en la visión por computadora y la inteligencia artificial. A pesar de su discontinuación en 2017, el Kinect vendió

8 millones de unidades en solo 60 días, dejando un legado perdurable en las tecnologías de interacción natural [20].

De manera similar, el Wii Remote de Nintendo, equipado con un acelerómetro que detecta la aceleración en tres ejes y un sensor infrarrojo que determina la posición relativa, brindó una experiencia de juego inmersiva y accesible. Utilizado en juegos emblemáticos como "Wii Sports" y complementado por el accesorio Nunchuk, el Wii Remote atrajo tanto a jugadores tradicionales como a un público más amplio, y también encontró aplicaciones en educación y rehabilitación física. El éxito del Wii Remote fue fundamental para la popularidad de la Wii, inspirando futuros desarrollos en la detección de movimiento dentro de la industria del entretenimiento interactivo [21].

Por su parte, el PlayStation Move, lanzado por Sony en septiembre de 2010 para la PlayStation 3 y compatible con la PlayStation 4 y PlayStation VR, utiliza controladores con sensores de aceleración y giroscopios, junto con una esfera luminosa rastreada por la PlayStation Eye/Camera, para detectar movimientos precisos en 3D. Su precisión y versatilidad mejoraron la jugabilidad en una amplia variedad de géneros y fueron esenciales para las experiencias de realidad virtual, consolidándose como un componente clave en la evolución de los controles de movimiento en los videojuegos [21].

1.5.4 Ambientes de Programación Processing, MATLAB y Arduino IDE

Processing es un entorno de desarrollo integrado (IDE) y lenguaje de programación flexible que se originó con el propósito de enseñar los fundamentos de la programación en un contexto visual y gráfico. Desarrollado inicialmente para artistas, diseñadores y educadores, Processing facilita la creación de gráficos interactivos, simulaciones, y visualizaciones de datos. Su sintaxis es simple y se basa en Java, lo que permite a los usuarios generar rápidamente prototipos visuales interactivos. El lenguaje y el entorno de Processing están diseñados para ser accesibles a personas

sin experiencia previa en programación, convirtiéndose en una herramienta esencial en campos como el diseño gráfico, la visualización de datos, la educación y el arte digital. Processing también se ha expandido a otras áreas, incluyendo la robótica y la electrónica, a través de bibliotecas especializadas que permiten la comunicación con hardware como microcontroladores y cámaras de movimiento [22], [23].

MATLAB es un entorno de programación de alto nivel y un entorno interactivo ampliamente utilizado en la ingeniería, las matemáticas, y la ciencia en general. Desarrollado por MathWorks, MATLAB ofrece una vasta gama de herramientas para la realización de cálculos numéricos, análisis de datos, simulaciones, y visualización gráfica. Una de las características más destacadas de MATLAB es su capacidad para manejar matrices y vectores de manera eficiente, lo que lo hace ideal para trabajos que involucran álgebra lineal, procesamiento de señales, y análisis estadístico. Además, MATLAB cuenta con una amplia colección de toolboxes que extienden sus capacidades a campos específicos como el procesamiento de imágenes, la inteligencia artificial, la simulación de sistemas dinámicos, y el diseño de controladores. En el contexto de la robótica, MATLAB se utiliza comúnmente para la modelación, simulación, y control de sistemas robóticos, así como para la implementación de algoritmos avanzados de procesamiento de señales y visión por computadora [24].

El Arduino IDE (Integrated Development Environment) es un entorno de programación diseñado específicamente para programar y depurar microcontroladores Arduino. Se caracteriza por su simplicidad y facilidad de uso, lo que ha contribuido a la popularidad de Arduino entre los aficionados, educadores y profesionales en el campo de la electrónica. El lenguaje de programación utilizado en el Arduino IDE está basado en C/C++, con una sintaxis simplificada que facilita el desarrollo de proyectos electrónicos interactivos. El entorno permite escribir, compilar y cargar código directamente en una amplia variedad de placas Arduino, lo que facilita

la implementación de proyectos que involucren sensores, actuadores y otros componentes electrónicos. El Arduino IDE también soporta la integración con diversas bibliotecas que extienden sus capacidades, permitiendo a los usuarios controlar dispositivos externos como motores, pantallas, y módulos de comunicación inalámbrica. En proyectos de robótica, el Arduino IDE se utiliza para desarrollar y controlar sistemas embebidos, donde la simplicidad y la flexibilidad son esenciales para el rápido prototipado y la iteración de diseños. [25].

1.5.5 Microprocesadores

Los microprocesadores son circuitos integrados que incluyen una CPU, memoria, puertos de entrada/salida y periféricos en un solo chip, diseñados para ejecutar tareas específicas en sistemas embebidos. Utilizados en una amplia gama de aplicaciones como electrónica de consumo, automoción, automatización industrial, medicina y hogares inteligentes, los microcontroladores permiten el control y la automatización eficiente. Ejemplos populares incluyen Arduino, PIC, AVR y STM32. Programados generalmente en C o ensamblador, ofrecen ventajas como integración, bajo consumo de energía y costo reducido, aunque tienen capacidades limitadas en comparación con los microprocesadores [26]. Los tipos de microcontroladores para usados para brazos robóticos son: Para el control de brazos robóticos, tres de los microcontroladores más utilizados son el STM32, el Arduino y el PIC. El STM32, basado en la arquitectura ARM Cortex, es valorado por su alto rendimiento y amplia gama de periféricos, siendo ideal para aplicaciones que requieren procesamiento intensivo y control preciso [27]. Arduino, con su plataforma de desarrollo fácil de usar y una gran comunidad de soporte, es popular en proyectos educativos y prototipos, ofreciendo flexibilidad y accesibilidad [28]. Los microcontroladores PIC, de Microchip Technology, son preferidos en entornos industriales debido a su robustez, fiabilidad y variedad de opciones, permitiendo un control eficiente y personalizado de los sistemas robóticos [29].

1.5.6 Mapeado del Cuerpo Humano Utilizando Cámaras de Movimiento

El mapeado del cuerpo humano mediante cámaras de movimiento es una tecnología avanzada que permite capturar y analizar el movimiento humano en tiempo real. Este proceso implica la detección y seguimiento de varios puntos clave del cuerpo, como las articulaciones y los segmentos corporales, utilizando sistemas de visión por computadora. Las cámaras de movimiento, que suelen estar equipadas con sensores infrarrojos y marcadores ópticos, capturan la posición y el movimiento de estos puntos en tres dimensiones, creando un modelo digital del cuerpo humano que puede ser utilizado para diversas aplicaciones, desde la animación digital hasta la rehabilitación física [31], [32].

El mapeado corporal se basa en algoritmos complejos de procesamiento de imágenes y análisis de movimiento, que transforman los datos capturados por las cámaras en representaciones precisas del cuerpo humano. Estos algoritmos son capaces de identificar y seguir marcadores colocados en puntos específicos del cuerpo o, en el caso de tecnologías más avanzadas, detectar características anatómicas directamente a través de cámaras sin necesidad de marcadores físicos. El resultado es un modelo tridimensional que refleja con alta fidelidad los movimientos y posturas del sujeto en tiempo real [33], [34].

Entre las aplicaciones más destacadas del mapeado corporal mediante cámaras de movimiento se encuentran la captura de movimientos para la creación de personajes en la industria del cine y los videojuegos, el análisis biomecánico en el ámbito deportivo y clínico, y el desarrollo de interfaces naturales para la interacción con sistemas digitales. En el contexto de la robótica, esta tecnología se utiliza para el control de robots humanoides y brazos robóticos, permitiendo una interacción más intuitiva y precisa entre el humano y la máquina. Al replicar los movimientos del operador humano, un brazo robótico puede realizar tareas complejas con un alto grado de destreza, mejorando la eficiencia y reduciendo los requisitos de capacitación especializada [5].

El avance en las cámaras de movimiento, como el desarrollo de sistemas sin marcadores y la integración de inteligencia artificial para el reconocimiento y predicción de movimientos, ha ampliado aún más las capacidades de mapeado del cuerpo humano. Estas tecnologías están evolucionando hacia soluciones más accesibles y precisas, facilitando su adopción en una variedad de campos y contribuyendo al desarrollo de nuevas aplicaciones que aprovechan el potencial del mapeado corporal en tiempo real.

1.5.7 Controladores

Los controladores en un sistema de control de brazo robótico son esenciales para asegurar movimientos precisos y seguros. Los controladores PID (Proporcional, Integral, Derivativo) son comúnmente utilizados para ajustar la posición y velocidad de las articulaciones basándose en el error entre la posición deseada y la actual, proporcionando un control preciso y estable. Además de los PID, se pueden utilizar controladores adaptativos que ajustan sus parámetros en tiempo real según las variaciones del sistema y controladores basados en modelos predictivos que anticipan el comportamiento futuro del sistema para optimizar las acciones actuales. Para tareas más complejas, como la manipulación de objetos y la interacción con el entorno, se pueden implementar algoritmos de control de alto nivel basados en inteligencia artificial y aprendizaje automático, que permiten al brazo robótico adaptarse y tomar decisiones en entornos dinámicos. La integración de estos controladores con los datos de los sensores garantiza que el brazo robótico pueda realizar tareas con precisión y eficiencia, respondiendo adecuadamente a cambios en el entorno y en las condiciones operativas [36]

Capítulo 2

2.1 Metodología.

En este capítulo se describe la metodología empleada para lograr los objetivos establecidos y encontrar la solución más adecuada al problema planteado. Se analizan los procesos llevados a cabo, que comprenden desde los requisitos de diseño y las posibles soluciones hasta los parámetros de diseño desde las perspectivas mecánica, electrónica y de control. Asimismo, se explican los conceptos y herramientas utilizados para alcanzar la solución final.

2.1.1 *Requerimientos de diseño*

Para lograr el diseño de un controlador para un brazo robótico mediante detección de movimiento, fue necesario tomar en cuenta cierto requerimientos los cuales se pueden apreciar en la **Tabla 2.1.**

Tabla 2.1

Requerimientos de diseño mecánico, eléctrico y de control

Tipo de Requerimiento	**Descripción del requerimiento**
Mecánico	• La base del brazo robótico a controlar debe proporcionarle estabilidad al momento de moverse. • Se deben emplear seis servomotores MG996R para dotar de movilidad al brazo. • El sistema en conjunto (robot y base) debe ser de fácil transporte para poder realizar las pruebas requeridas en distintos espacios abiertos.
Electrónico	• El sistema debe poder ser alimentado mediante el uso de cualquier toma de corriente de 110V.

De control	• El robot deberá seguir en tiempo real los movimientos de un usuario mediante la recopilación de datos usando una cámara de movimiento. • El movimiento del robot debe realizarse evitando cualquier tipo de colisiones consigo mismo o la estructura de la base. • Los movimientos realizados por el robot deben ser precisos, traduciendo de la mejor forma posible el movimiento capturado del usuario. • Para el proceso de captura de datos debe usarse una Kinect v1.

2.1.2 *Selección de alternativas de solución*

Para poder resolver la problemática abordada en el Capítulo I, se consideraron dos aspectos fundamentales para los cuales se tendrían diversas alternativas de solución. Estos aspectos mencionados son la librería de captura/procesamiento de datos y el tipo de control, donde las alternativas de cada uno de ellos se detallarán en cada sección.

2.1.2.1 Librería de captura/procesamiento de datos. Para este apartado se tenían a disposición diversas herramientas dedicadas a la detección de movimiento y mapeado del cuerpo humano mediante la utilización de Kinect, tanto en su versión 1 como 2. Por ello, se consideraron tres principales alternativas que podían ayudarnos a desarrollar el proyecto planteado:

- **Alternativa uno:** Utilización de librería SimpleOpenNI en el ambiente Processing. Para desarrollar un controlador para un brazo robótico mediante la detección de movimiento usando una cámara y la librería SimpleOpenNI en Processing, se comenzaría por utilizar la cámara para capturar los movimientos de un usuario. La librería SimpleOpenNI, previamente instalada en Proccesing, se encargaría de interpretar estos movimientos,

identificando las posiciones de las articulaciones relevantes. Estas posiciones se traducirían en coordenadas que se mapean a movimientos específicos del brazo robótico. Posteriormente, se enviarían comandos al brazo robótico para replicar los movimientos detectados por la cámara, permitiendo que el brazo se controle en tiempo real a través de la detección de movimiento del usuario [42].

- **Alternativa dos:** Utilización del kit de desarrollo Kinect SDK en conjunto a OpenCV. Para desarrollar un controlador para un brazo robótico utilizando OpenCV y Kinect SDK, se emplearía la Kinect para capturar y analizar los movimientos del usuario. El Kinect SDK se encargaría de proporcionar los datos de profundidad y posición de las articulaciones del cuerpo. Luego, OpenCV procesaría estas imágenes para extraer las coordenadas exactas de las articulaciones. Estas coordenadas se convertirían en comandos que controlan el brazo robótico, mapeando los movimientos del usuario a movimientos precisos del brazo. Los comandos resultantes se enviarían al controlador del brazo robótico, permitiendo que el brazo reproduzca en tiempo real los movimientos detectados por la Kinect. Sin embargo, su uso podría implicar una mayor complejidad en el desarrollo, debido a la necesidad de implementar algoritmos de procesamiento y gestionar la integración de software, lo cual podría aumentar el tiempo de desarrollo y los desafíos técnicos [44].

- **Alternativa tres:** Utilización PCL (Point Cloud Library). Esta opción permite procesar nubes de puntos obtenidas de la Kinect. PCL proporcionaría herramientas avanzadas para la manipulación y análisis de datos 3D, permitiendo la reconstrucción detallada de modelos tridimensionales y el análisis espacial, lo que luego se podría traducir en comandos específicos para la articulación del brazo robótico. Aunque esta alternativa sería útil para un análisis profundo del entorno 3D, su uso puede ser más técnico y complejo debido a la

necesidad de gestionar grandes volúmenes de datos de nubes de puntos y realizar un procesamiento avanzado [45].

2.1.2.1.1 Criterios de selección de solución (Librería de Procesamiento). Para poder escoger la mejor alternativa, se tomaron en consideración cuatro criterios de evaluación, los cuales se describirían de la siguiente manera:

- **A: Soporte de la comunidad.** Este apartado se refiere a la disponibilidad y calidad de la ayuda y los recursos ofrecidos por la comunidad de usuarios y desarrolladores. Esto incluye foros, grupos de discusión, documentación, tutoriales y respuestas a preguntas frecuentes, que facilitan resolver problemas y aprender a usar la librería de manera efectiva.
- **B: Capacidad de traducción de lecturas.** Este apartado se basa en la habilidad de las herramientas que disponga la librería para poder convertir datos obtenidos de la lectura de movimiento en datos numéricos con los que se pueda desarrollar un intercambio entre la parte mecánica y de control.
- **C: Complejidad de desarrollo.** Este criterio trata sobre la dificultad implícita en el desarrollo tanto de la toma de datos como del procesamiento de estos usando cada una de las librerías. Esta dificultad radica en el volumen de datos recopilados, filtración de información innecesaria y existencia de funciones que agilicen estos procesos.
- **D: Capacidad de extracción de datos.** Este criterio radica en la habilidad de cada librería para poder obtener diferentes datos sobre una lectura en tiempo real en concreto. Esta capacidad se ve reflejada en cantidad de puntos de referencias corporales detectados, articulaciones preestablecidas y herramientas de manejo de datos.

2.1.2.1.2 Matriz de decisión (Librería de Procesamiento). Habiendo establecido estos criterios de selección, se pudo construir nuestra matriz de decisión que nos llevó a seleccionar la mejor alternativa de librería para procesar los datos.

Tabla 2.2

Peso de cada uno de los criterios seleccionados

	Pesos de los criterios					
Criterios	**A**	**B**	**C**	**D**	**Σ+1**	**Ponderación**
A		0	0	0	1	0,1
B	1		1	0,5	3,5	0,35
C	1	0		0	2	0,2
D	1	0,5	1		3,5	0,35
				Suma	10	1

Tabla 2.3

Ponderación de cada alternativa con respecto al criterio A

	Criterio A: Soporte de la comunidad				
Soluciones	**1**	**2**	**3**	**Σ+1**	**Ponderación**
1		0	0	1	0,166666667
2	1		1	3	0,5
3	1	0		2	0,333333333
			Suma	6	1

SimpleOpenNI ofrecía un soporte más limitado, con una comunidad menos activa y menos recursos disponibles. OpenCV, en cambio, contaba con un soporte amplio, respaldado por una gran comunidad y una extensa documentación y recursos. Por su parte, PCL (Point Cloud Library) disponía de un buen soporte, con una comunidad activa y recursos adecuados, aunque está más especializado en el manejo de nubes de puntos, lo que podía limitar la cantidad de soporte en comparación con OpenCV.

Tabla 2.4

Ponderación de cada alternativa con respecto al criterio B

	Criterio B: Capacidad de traducción de lecturas				
Soluciones	**1**	**2**	**3**	**Σ+1**	**Ponderación**
1		1	1	3	0,5
2	0		1	2	0,333333333

3	0	0	—	1	0,166666667
			Suma	6	1

SimpleOpenNI proporcionaba funciones directas para traducir lecturas de la Kinect en datos de posición y esqueleto. OpenCV requería integración adicional con el SDK de Kinect para procesar datos de profundidad e imágenes, ofreciendo flexibilidad para el análisis de imágenes, pero sin funciones específicas para Kinect. PCL manejaba datos de nubes de puntos y requería un procesamiento adicional para traducir lecturas de Kinect en información útil sobre la forma y estructura del entorno.

Tabla 2.5

Ponderación de cada alternativa con respecto al criterio C

	Criterio C: Complejidad de desarrollo				
Soluciones	**1**	**2**	**3**	**Σ+1**	**Ponderación**
1	—	1	1	3	0,5
2	0	—	1	2	0,333333333
3	0	0	—	1	0,166666667
			Suma	6	1

SimpleOpenNI era la más sencilla, con una API enfocada en la integración directa con Kinect. OpenCV era más compleja, ofreciendo herramientas versátiles para el procesamiento de imágenes, pero requiriendo integración adicional para Kinect. PCL era la más compleja, especializada en nubes de puntos y con una curva de aprendizaje pronunciada para el manejo y análisis de datos 3D.

Tabla 2.6

Ponderación de cada alternativa con respecto al criterio D

	Criterio D: Capacidad de extracción de datos				
Soluciones	**1**	**2**	**3**	**Σ+1**	**Ponderación**
1	—	0,5	0,5	2	0,333333333
2	0,5	—	1	2,5	0,416666667
3	0,5	0	—	1,5	0,25
			Suma	6	1

SimpleOpenNI permitía una extracción de datos de imágenes en tiempo real específica para Kinect, con facilidad de uso. OpenCV ofrecía potente extracción y procesamiento de imágenes en tiempo real, pero requiere integración con otros SDKs para Kinect. PCL se enfocaba en nubes de puntos en tiempo real, con un enfoque menos directo para datos de imágenes 2D.

Tabla 2.7

Ponderaciones finales de todos los criterios

	Conclusiones					
Soluciones	**A*0,1**	**B*0,35**	**C*0,2**	**D*0,35**	**Suma**	**Prioridad**
1	0,016667	0,175	0,1	0,116667	0,408333	1
2	0,05	0,116667	0,066667	0,145833	0,379167	2
3	0,033333	0,058333	0,033333	0,0875	0,2125	3

En conclusión, se optó por usar la librería SimpleOpenNI, ya que satisfacía de mejor manera cada uno de los cuatro apartados tomados en consideración.

2.1.2.2 Tipo de control. Para este apartado evaluamos cuatro posibles métodos de control que podíamos utilizar para el manejo del brazo robótico:

- **Alternativa uno:** Control por posición. El control por posición para un brazo robótico utilizando una cámara de movimiento implica captar la posición del usuario mediante la cámara, procesar estas imágenes para identificar las coordenadas específicas, y convertir estas coordenadas en comandos de movimiento para el brazo robótico. El brazo se ajusta en tiempo real para replicar las posiciones detectadas por la cámara, permitiendo una manipulación precisa y sincronizada con los movimientos del usuario [46].
- **Alternativa dos:** Control por velocidad. Por otro lado, este control captura la velocidad de movimiento del usuario a través de la cámara. Estos datos de velocidad se procesan para determinar cómo debe moverse el brazo robótico. Los comandos de velocidad se envían al brazo, ajustando su movimiento en tiempo real para seguir la rapidez y dirección del usuario, logrando un control dinámico y fluido [47].

- **Alternativa tres:** Control por fuerza Este control radica en la estimación de la fuerza aplicada por el usuario a través de la cámara, analizando cómo varía su postura o movimiento. Esta información se procesa para ajustar la fuerza ejercida por el brazo robótico, permitiendo que el brazo reproduzca las intensidades de fuerza del usuario en tiempo real, logrando un control preciso y adaptativo basado en la fuerza detectada [48].
- **Alternativa cuatro:** Control por impedancia En este tipo de control, se ajusta la rigidez y la respuesta del brazo en función de las posiciones y movimientos del usuario. La cámara detecta cómo se mueve el usuario y la variabilidad de su postura. Estos datos se usan para adaptar la impedancia del brazo robótico, regulando su resistencia y flexibilidad para que responda de manera similar a las fuerzas y movimientos aplicados por el usuario, proporcionando un control más natural y adaptable [49].

2.1.2.2.1 ***Criterios de selección de solución (Librería de Procesamiento).*** Para poder escoger la mejor alternativa, se tomaron en consideración nuevamente cuatro criterios de evaluación, pero enfocados al control:

- **A: Estabilidad.** Este criterio se refiere a la capacidad del sistema para mantener un comportamiento constante y predecible sin oscilaciones o divergencias en la respuesta.
- **B: Precisión.** Este apartado se basa en a la capacidad del sistema para alcanzar y mantener el valor deseado con exactitud.
- **C: Rapidez de respuesta.** Este criterio radica en la velocidad con la que el sistema reacciona a cambios o comandos.
- **D: Robustez.** Este apartado habla sobre la capacidad del sistema para mantener su desempeño frente a variaciones o perturbaciones inesperadas.

2.1.2.2.1 Matriz de decisión (Tipo de control). Una vez definidos estos nuevos criterios de selección, se elaboró la respectiva matriz de decisión para dar con el tipo de control más beneficioso para el proyecto.

Tabla 2.8

Peso de cada uno de los criterios seleccionados

	Pesos de los criterios					
Criterios	**A**	**B**	**C**	**D**	**Σ+1**	**Ponderación**
A	—	0	0	0	1	0,1
B	1	—	1	0,5	3,5	0,35
C	1	0	—	1	3	0,3
D	1	0,5	0	—	2,5	0,25
				Suma	10	1

Tabla 2.9

Ponderación de cada alternativa con respecto al criterio A

	Criterio A: Estabilidad					
Soluciones	**1**	**2**	**3**	**4**	**Σ+1**	**Ponderación**
1	—	1	0,5	0	2,5	0,25
2	0	—	0	0	1	0,1
3	0,5	1	—	0	2,5	0,25
4	1	1	1	—	4	0,4
				Suma	10	1

El control por posición tiende a ser relativamente estable, pero podría experimentar oscilaciones si el sistema está sujeto a perturbaciones o errores de medición, similar a lo que sucede con el control por fuerza. Por su parte, el control por impedancia proporcionaba una estabilidad robusta al adaptar la rigidez y flexibilidad del sistema, lo que permite manejar perturbaciones y cambios en el entorno de manera efectiva.

Tabla 2.10

Ponderación de cada alternativa con respecto al criterio B

	Criterio B: Precisión					
Soluciones	**1**	**2**	**3**	**4**	**Σ+1**	**Ponderación**
1	—	1	1	1	4	0,4
2	0	—	0	0	1	0,1
3	0	1	—	0	2	0,2
4	0	1	1	—	3	0,3

				Suma	10	1

En cuanto a precisión, el control por posición ofrecía alta precisión, permitiendo movimientos exactos hacia ubicaciones específicas gracias a su enfoque en el ajuste preciso de la posición mientras que el control por velocidad podía ser menos preciso en comparación, ya que se centraba en mantener una velocidad constante y podía tener dificultades para ajustar la posición con exactitud, Así mismo, el control por fuerza proporcionaba buena precisión en la aplicación de fuerzas, pero podía ser menos exacto en términos de ubicación precisa debido a su enfoque en la fuerza aplicada. Por último, el control por impedancia combinaba flexibilidad y rigidez, lo que podía impactar en la precisión, ya que adaptaba tanto la rigidez como la flexibilidad para manejar perturbaciones, a veces sacrificando precisión en movimientos exactos.

Tabla 2.11

Ponderación de cada alternativa con respecto al criterio C

	Criterio C: Rapidez de respuesta					
Soluciones	**1**	**2**	**3**	**4**	**Σ+1**	**Ponderación**
1		0	1	1	3	0,3
2	1		1	1	4	0,4
3	0	0		1	2	0,2
4	0	0	0		1	0,1
				Suma	10	1

Para el criterio C, el control por posición tenía una velocidad de respuesta moderada, ajustando la posición del brazo robótico con cierta latencia dependiendo de la complejidad del movimiento. El control por velocidad destacaba por su rápida velocidad de respuesta, ya que se enfocaba en mantener una velocidad constante y ajustar rápidamente el movimiento. En contraste, el control por impedancia, aunque era flexible y se ajustaba bien a las perturbaciones, podía tener una velocidad de respuesta más lenta debido a la necesidad de ajustar tanto la rigidez como la flexibilidad del sistema.

Tabla 2.12

Ponderación de cada alternativa con respecto al criterio D

	Criterio D: Robustez					
Soluciones	**1**	**2**	**3**	**4**	**Σ+1**	**Ponderación**
1		0	0	0	1	0,1
2	1		0	0	2	0,2
3	1	1		0,5	3,5	0,35
4	1	1	0,5		3,5	0,35
				Suma	10	1

El control por fuerza se destacaba por su alta robustez, ya que estaba diseñado para manejar perturbaciones ajustando la fuerza aplicada, lo que aseguraba una respuesta estable en condiciones cambiantes. De igual forma, el control por impedancia ofrecía una robustez igualmente alta, adaptando la rigidez y flexibilidad del sistema para manejar perturbaciones y variaciones en el entorno, proporcionando así una respuesta flexible y estable.

Tabla 2.13

Ponderaciones finales de todos los criterios

	Conclusiones					
Soluciones	**A*0,1**	**B*0,35**	**C*0,3**	**D*0,25**	**Suma**	**Prioridad**
1	0,025	0,14	0,09	0,025	0,28	1
2	0,01	0,035	0,12	0,05	0,215	4
3	0,025	0,07	0,06	0,0875	0,2425	3
4	0,04	0,105	0,03	0,0875	0,2625	2

Como conclusión, se optó por el control de posición, ya que este tipo de control se adaptaba mejor a la aplicación específica, satisfaciendo de manera óptima los cuatro criterios evaluados.

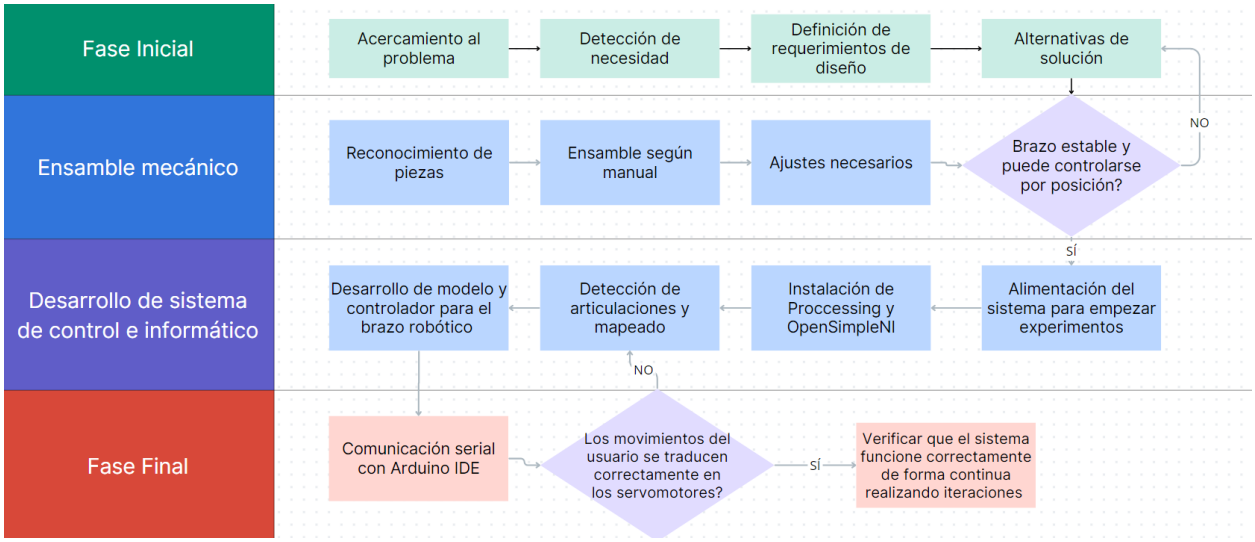
2.1.3 *Proceso de diseño*

Una vez definidas las mejores propuestas capaces de cumplir con los requerimientos del proyecto, se estableció un diagrama de flujo de trabajo, determinando el paso a paso para establecer cada una de las fases de desarrollo.

En la **Figura 2.1** se puede observar que el flujo de trabajo es casi lineal, exceptuando por ciertas iteraciones que se deberían realizar en caso de que no se cumpla con la condición dada.

Figura 2.1

Flujo de trabajo para desarrollar controlador de brazo robótico mediante cámara de movimiento

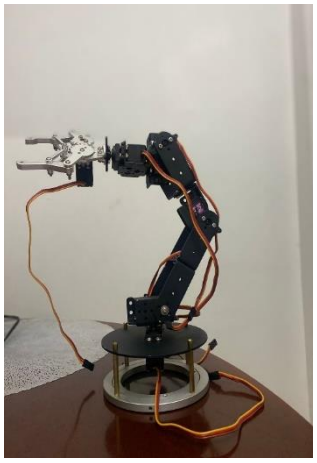


2.1.4 *Diseño*

2.1.4.1 ***Diseño mecánico.*** En cuanto al diseño mecánico del proyecto, cabe destacar que el cliente impuso un kit de ensamble específico de un brazo robótico de seis grados de libertad. Este robot contaba con seis servomotores de alto torque MG996R y un efector final en forma de garra, la cual era accionada por dos engranajes. El ensamble completo del brazo se puede apreciar con facilidad en la **Figura 2.2**, donde se evidencia el modelo escogido para el proyecto.

Figura 2.2

Brazo robótico ensamblado



En ciertas ocasiones, fue necesario realizar modificaciones al diseño de ensamble de fábrica, ya que se tuvieron que realizar modificaciones en los acoples de los servomotores para que encajaran de mejor manera con los eslabones del brazo robótico.

Otro aspecto crucial fue calibrar la posición de los motores para evitar colisiones durante el movimiento del brazo. El rango de movimiento de estos servomotores iba de 0 a 180 grados, por lo que tuvimos que definir que grados eran operacionales sin que el robot colisionara contra sí mismo, los cuales se pueden apreciar en la **Tabla 2.14**.

Tabla 2.14

Ángulos de trabajo de cada servomotor donde se evitan colisiones

Articulación	**Ángulos que evitan colisión**
1	Desde 0 hasta 180
2	Desde 10 hasta 150
3	Desde 10 hasta 160
4	Desde 20 hasta 160
5	Desde 0 hasta 180
6	Desde 0 hasta 180

2.1.4.2 ***Diseño de control.*** El desarrollo del controlador se llevó a cabo siguiendo una serie de pasos detallados que aseguraron la correcta implementación y funcionamiento del proyecto.

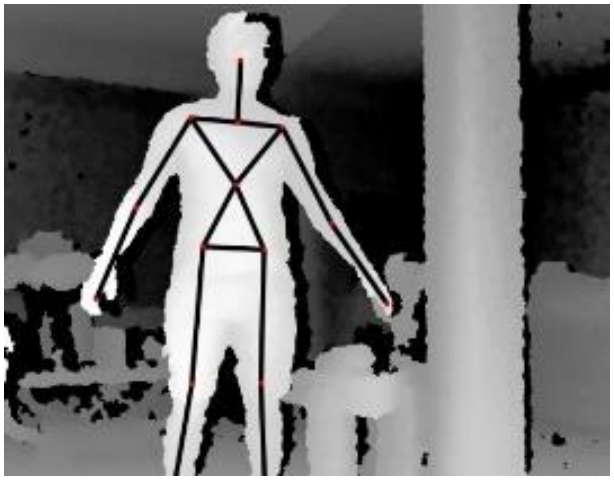
Inicialmente, se procedió a la configuración del entorno de desarrollo. Esto incluyó la instalación del software necesario, comenzando por el Processing IDE, el Arduino IDE, y los drivers y SDK necesarios para la Kinect 360. Además, se instaló la librería SimpleOpenNI en Processing para permitir la interacción con la Kinect. La conexión del hardware implicó conectar la Kinect 360 a la computadora mediante un cable USB y los servomotores al microcontrolador

Arduino, asegurando la correcta disposición y funcionamiento de todos los componentes. Se verificó que tanto la Kinect como los servomotores estaban operativos antes de proceder con el desarrollo del controlador.

El siguiente paso consistió en la adquisición de ángulos corporales usando Processing. Se inició con la configuración de la librería SimpleOpenNI para inicializar la Kinect y habilitar las funcionalidades de profundidad y detección de usuarios. Se verificó que la Kinect detecte correctamente a los usuarios y sus articulaciones. A continuación, se implementó la captura de datos del esqueleto utilizando los métodos proporcionados por la librería SimpleOpenNI para detectar las posiciones tridimensionales de las articulaciones relevantes, tales como la cabeza, manos, codos y hombros tal como lo muestra la **Figura 2.3**. Estos valores luego nos sirvieron como señal de entrada para los movimientos del brazo robótico. Una vez asegurado que los datos se capturaban de manera correcta en tiempo real, se desarrolló una función para calcular los ángulos entre las articulaciones. Esta función utilizaba la fórmula del producto punto e identidades trigonométricas para determinar los ángulos con precisión.

Figura 2.3

Detección de esqueleto y articulaciones de un usuario mediante Processing



Una vez calculados los ángulos, se procedió a crear el modelo del brazo robótico de manera experimental. Para esto se hicieron experimentos por cada una de las articulaciones donde se recorría un ángulo operativo por cada segundo que pasaba. Es decir, para la articulación 1 se recorrió del ángulo 0 a 180, tomando una total de 181 segundos, donde disponíamos del ángulo escrito y el ángulo en el que realmente se posicionaba el servomotor. Esto se realizó con las demás articulaciones posicionadas en los ángulos medios de su rango operacional.

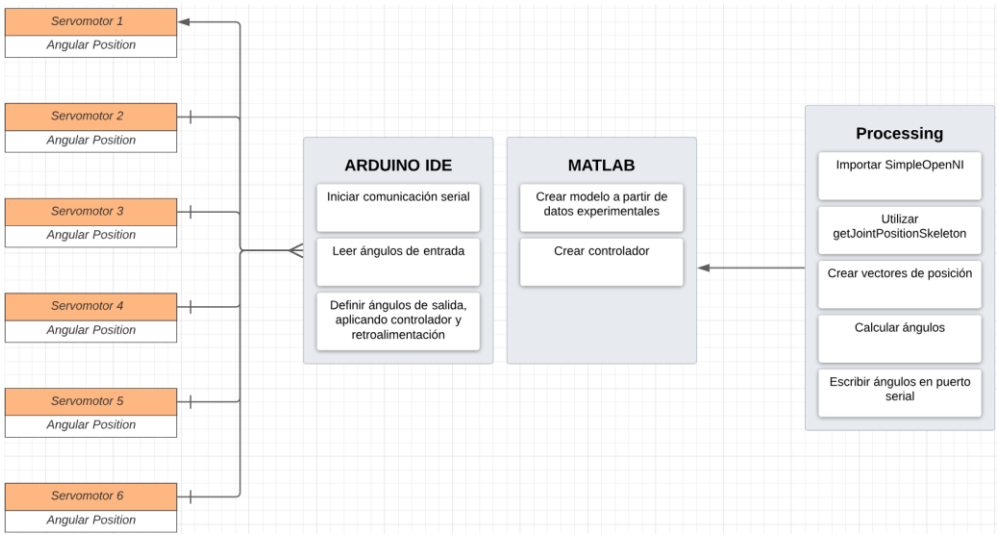
Luego, se construyó un modelo utilizando MATLAB y su función de mínimos cuadrados a partir de dos arreglos: uno con los ángulos impuestos al servomotor y otro con su posición real. Este modelo luego se pasó a una función de transferencia que fue la que nos ayudó a visualizar la respuesta de este.

Después de ello, teniendo el modelo experimental del comportamiento de cada articulación, se utilizó el sintonizador automático de parámetros Kp y Ki de MATLAB para mejorar las respuestas en cada articulación. De manera manual, se ajustaron estos valores en ejercicios iterativos hasta percibir un comportamiento deseado en la respuesta del sistema.

Finalmente, se pasaron los ángulos obtenidos desde la cámara de movimiento por Processing a través del controlador desarrollado en MATLAB hacia Arduino IDE por el puerto serial para que estos valores sean asignados a los servomotores, estableciendo una retroalimentación negativa en esta última plataforma de desarrollo para disminuir el error de estado estable. En la **Figura 2.4** se puede apreciar la interacción de estas plataformas de desarrollo.

Figura 2.4

Interacción entre Processing, MATLAB y Arduino IDE



2.1.4.3 ***Diseño eléctrico.*** Para el desarrollo del sistema eléctrico fue necesaria la implementación de dos adaptadores de voltaje de 5 Voltios a 2 Amperios para poder suministrar energía suficiente para que ningún servomotor tuviera perdidas de torque o un sobrecalentamiento.

2.1.5 *Consideraciones Éticas y Legales*

El diseño e implementación del sistema tomaron en cuenta diversas consideraciones éticas y legales cruciales, especialmente debido al uso de datos biométricos y la posible aplicación en contextos sensibles. La privacidad y protección de datos fueron prioridades, garantizando que la captura de datos del esqueleto humano mediante el Kinect se realizara con fines específicos del proyecto y que estos datos no se almacenaran más allá del tiempo necesario. Se aseguró que los usuarios proporcionaran su consentimiento informado antes de participar, entendiendo claramente qué datos se recogían, cómo se procesaban y para qué se utilizaban. Además, se consideró el impacto potencial de la tecnología en la privacidad y seguridad de los usuarios, garantizando que se le dará únicamente un uso ético y responsable del sistema.

Capítulo 3

3.1 Resultados y análisis

En esta sección se detallarán los resultados obtenidos para el desarrollo del controlador y durante el funcionamiento del brazo robótico.

3.1.1 Diseño del controlador para articulaciones del robot

A continuación, se presentarán los sistemas identificados y las respuestas correspondientes para cada una de las articulaciones del robot. Es importante destacar que, para la identificación de cada sistema, se empleó el método de mínimos cuadrados. Esta elección se basó en la disponibilidad de conjuntos de datos relativamente pequeños, donde la robustez inherente de este método permitió obtener una estimación precisa y confiable. Una vez creados los modelos, a través de Matlab se obtuvieron las respuestas en escalón de cada una de las funciones de transferencia.

Figura 3.1

Respuesta escalón del sistema de la articulación 1

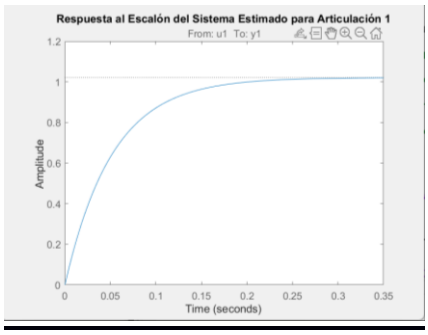


Figura 3.2

Respuesta escalón del sistema de la articulación 2

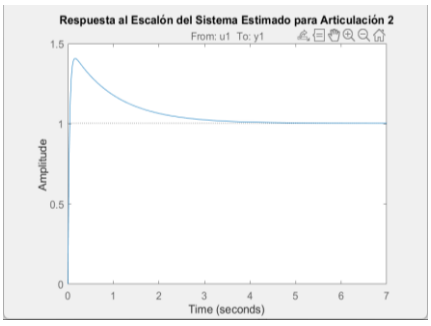


Figura 3.3

Respuesta escalón del sistema de la articulación 3

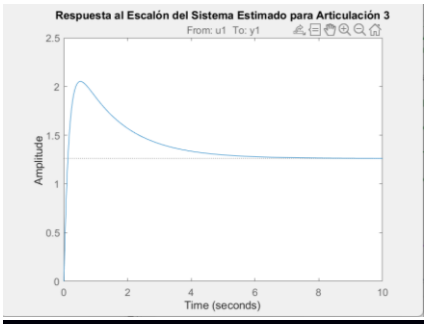


Figura 3.4

Respuesta escalón del sistema de la articulación 4

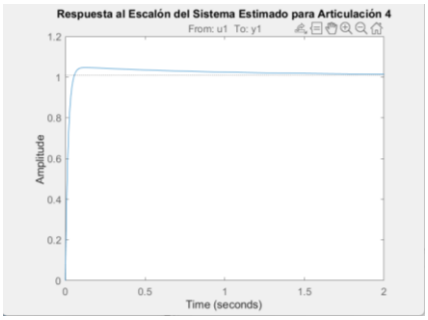


Figura 3.5

Respuesta escalón del sistema de la articulación 5

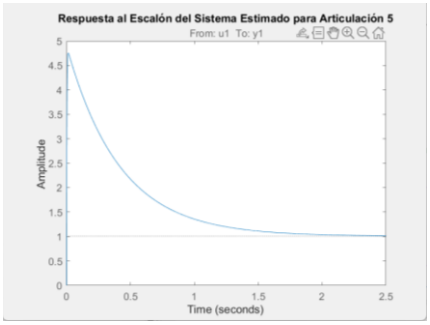
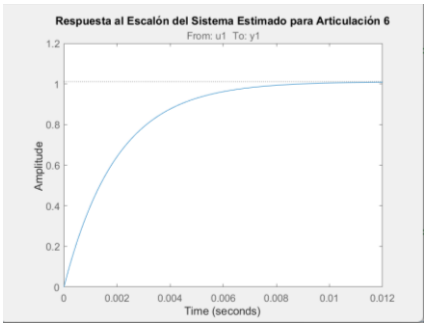


Figura 3.6

Respuesta escalón del sistema de la articulación 6



Como se puede observar en las gráficas presentadas, no todas las articulaciones muestran comportamientos erráticos, como se evidencia en la **Figuras 3.1** y **Figura 3.6**, donde los sistemas presentan un sobre amortiguamiento sin complicaciones adicionales. Sin embargo, esta situación no es uniforme en todas las respuestas analizadas. En la **Figura 3.2**, se observa que el sistema experimenta un sobre impulso considerable y un tiempo de estabilización elevado, lo cual es problemático considerando la naturaleza del proyecto, que requiere un seguimiento articular en tiempo real. En la **Figura 3.3**, se puede apreciar un comportamiento más errático, donde el sistema alcanza una amplitud de hasta 2 unidades, lo que resulta en un sobre impulso significativo y un tiempo de estabilización de 10 segundos. Al analizar la **Figura 3.4**, se nota que el sistema exhibe un tiempo de estabilización algo prolongado, pero con un nivel de sobre impulso bajo, lo que permite que el movimiento sea coherente con los datos capturados. Finalmente, en la **Figura 3.5** se observa un pico que excede considerablemente el valor esperado. A pesar de que el tiempo de estabilización es inferior a medio segundo, es necesario corregir el sobre impulso excesivo para mejorar el rendimiento.

Para abordar estos desafíos, se sintonizaron automáticamente las ganancias Kp y Ki utilizando la función *pidtune* en Matlab, basándose en las respuestas escalón de cada función de transferencia de las articulaciones. Esta decisión se tomó después de intentar aplicar el método de Ziegler-Nichols mediante la obtención de la ganancia y el período críticos, pero se encontró que

la estimación de estos parámetros requería un procesamiento computacional considerable, el cual no estaba disponible en los equipos de prueba. En la **Tabla 3.1** se presentan los valores correspondientes a cada uno de estos parámetros.

Tabla 3.1

Valores de parámetros Kp y Ki calculados para cada articulación

Articulación	**Parámetro**	**Valor**
1	Kp	1.4026
	Ki	48.8572
2	Kp	1.4904
	Ki	14.70
3	Kp	1.6450
	Ki	20.3192
4	Kp	1.1560
	Ki	20.4142
5	Kp	1.1763
	Ki	-30.778
6	Kp	1.4159
	Ki	12.934

Con los valores de los parámetros obtenidos, se construyó un sistema en lazo cerrado con retroalimentación unitaria para verificar el funcionamiento de dichos valores. Al igual que en las primeras gráficas de respuesta del sistema, se empleó una entrada de tipo escalón para analizar y evidenciar el comportamiento del sistema bajo las nuevas condiciones. Esta metodología permite evaluar de manera directa cómo los parámetros sintonizados influyen en la estabilidad y el

rendimiento del sistema, asegurando que el control implementado sea adecuado para las exigencias operativas del proyecto.

Figura 3.7

Respuesta escalón del sistema retroalimentado PI de la articulación 1

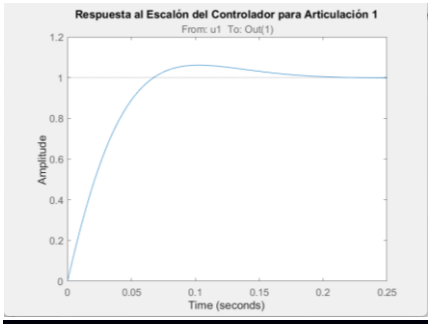


Figura 3.8

Respuesta escalón del sistema retroalimentado PI de la articulación 2

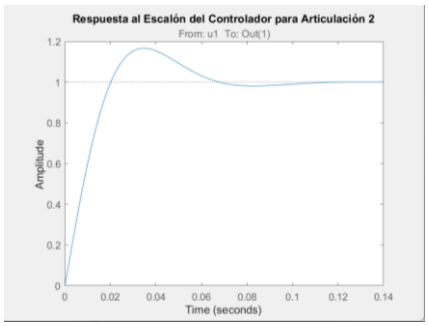


Figura 3.9

Respuesta escalón del sistema retroalimentado PI de la articulación 3

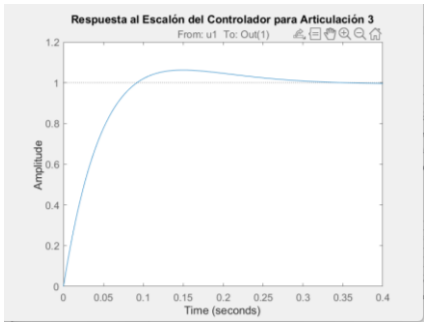


Figura 3.10

Respuesta escalón del sistema retroalimentado PI de la articulación 4

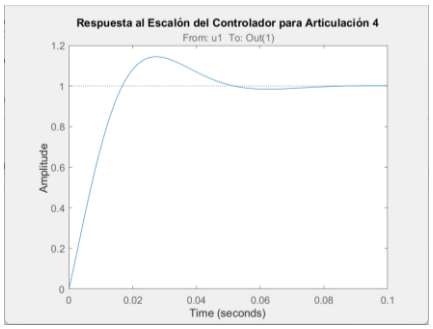


Figura 3.11

Respuesta escalón del sistema retroalimentado PI de la articulación 5

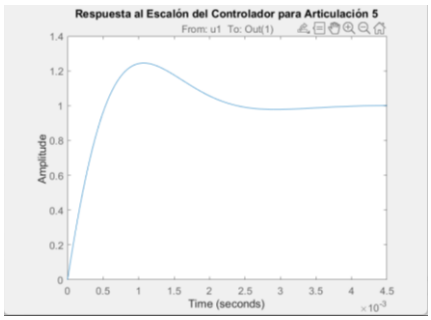
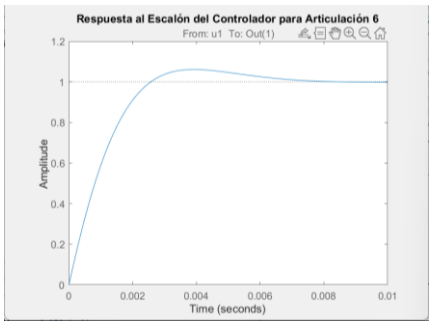


Figura 3.12

Respuesta escalón del sistema retroalimentado PI de la articulación 6



De acuerdo con las gráficas obtenidas, se puede observar que todas las respuestas presentan ahora tiempos de estabilización inferiores a 0.4 segundos, lo cual es especialmente adecuado para el seguimiento en tiempo real de los movimientos articulares de una persona. En la **Figura 3.7**, se evidencia que el sobre amortiguamiento presente anteriormente fue eliminado, y el tiempo de estabilización se redujo en 0.10 segundos. Aunque persiste un ligero sobre impulso, este podría ser eliminado con un afinamiento más exhaustivo de los parámetros. En la **Figura 3.8**, se destaca una notable mejora no solo por la reducción del sobre impulso en aproximadamente un 20%, sino

también porque se logró disminuir el tiempo de estabilización a tan solo 0.14 segundos, lo que representa una reducción de 6.86 segundos en comparación con el valor anterior. La **Figura 3.9** muestra una mejoría aún más significativa, con una reducción del sobre impulso de aproximadamente un 90% y un tiempo de estabilización actual de 0.4 segundos, es decir, 9.6 segundos menor que el registrado previamente. La **Figura 3.10** revela que, aunque el sistema ha ganado aproximadamente un 8% de sobre impulso en comparación con el sistema inicial, se observa una mejoría en el tiempo de estabilización, que se reduce de 2 segundos a 0.1 segundos, lo que representa una diferencia de 1.9 segundos. Por otro lado, la **Figura 3.11** presenta mejoras sustanciales tanto en la disminución del sobre impulso como en el tiempo de estabilización. En cuanto al sobre impulso, se reduce en un 355%, eliminando así el movimiento errático que se producía en esta articulación al escribir un nuevo ángulo en el servomotor. En términos de tiempo de estabilización, la mejora también es notable, reduciéndose de 2.5 segundos a tan solo 0.0045 segundos, siendo este el sistema más rápido de todos. Finalmente, la **Figura 3.12** muestra que el sistema ha eliminado el sobre amortiguamiento, aunque ha ganado un 4% de sobre impulso. Sin embargo, se aprecia una ligera mejoría en el tiempo de estabilización, que se reduce de 0.012 a 0.010 segundos, una diferencia que, aunque pequeña, es una indicación positiva de la mejora en el rendimiento del sistema, aunque imperceptible a simple vista.

3.1.2 Funcionamiento del brazo robótico con el controlador integrado

Una vez definidas las ganancias que optimizarían el movimiento articular del robot, se procedió a implementar cada una de ellas en el entorno de Arduino IDE, donde funcionarían en tiempo real en conjunto con los ángulos de las articulaciones de un usuario, obtenidos a través del dispositivo Kinect y procesados en el entorno de Processing. Estas posiciones angulares se transmitieron mediante comunicación serial al entorno de Arduino, y utilizando la librería *PID_v1*, se empleó el fragmento de código mostrado en la **Figura 3.13** para instanciar el controlador

encargado de eliminar los errores posicionales y reducir las vibraciones en el sistema. Este enfoque permitió que el robot replicara de manera precisa los movimientos del usuario, asegurando una operación estable y eficiente.

Figura 3.13

Código de Arduino IDE que permite crear controladores para cada articulación con los parámetros Kp y Ki previamente obtenidos

```
for (int i = 0; i < 6; i++) {
  servos[i].attach(servoPins[i]);
  pidControllers[i] = PID(&input[i], &output[i], &setpoint[i], Kp[i], Ki[i], DIRECT);
  pidControllers[i].SetMode(AUTOMATIC);
  pidControllers[i].SetOutputLimits(0, 180); // Limitar la salida a 0-180 grados
}
```

En este fragmento de código se puede apreciar los arreglos input, output, setPoint, Kp y Ki para definir los controladores. Estos arreglos vienen dados por los ángulos actuales de los servomotores, las señales de control de retroalimentación para el PI, los ángulos deseados en grados, las ganancias Kp y las ganancias Ki obtenidas anteriormente desde Matlab. En la **Figura 3.14** y **Figura 3.15**, es posible visualizar el funcionamiento del brazo robótico en conjunto al sistema de lectura de datos. En este ejemplo se definió seis articulaciones para controlar el movimiento articular del brazo robótico: los hombros, codos y rodillas en donde el hombro derecho controla la articulación 1, el codo derecho la articulación 2, el hombro izquierdo la articulación 3, el codo izquierdo la articulación 4, la rodilla derecha la articulación 5 y la rodilla izquierda se encarga de la articulación 6.

Figura 3.14

Prueba 1 del funcionamiento del brazo robótico con ambiente de captura de movimiento

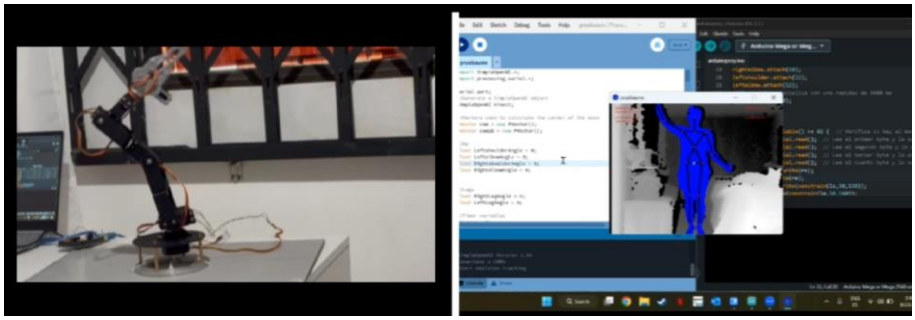
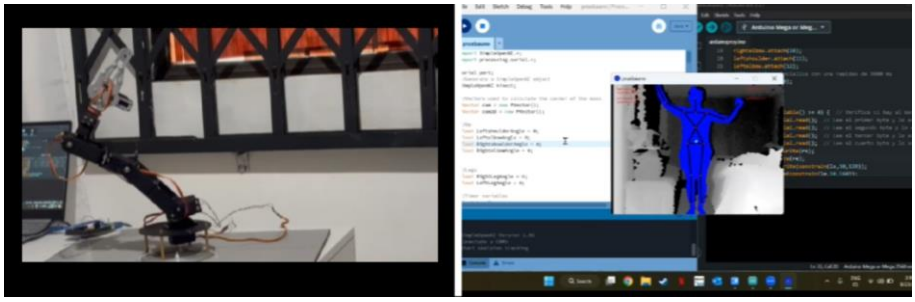


Figura 3.15

Prueba 2 del funcionamiento del brazo robótico con ambiente de captura de movimiento



3.2 Análisis de costos

Debido a la arquitectura del proyecto, para llevarlo a cabo se necesita de una estructura física, eléctrica e informática. Es por ello que para la evaluación de los costos totales hemos incluido tanto costos de elementos físicos como de licencias de software necesarias para realizar el procesamiento de datos.

Comenzando por el apartado físico, el brazo robótico utilizado puede ser encontrado en páginas de comercio libre como AliExpress desde un valor de $40. Para la estructura que se utiliza para estabilizar la base, se adquirió una plancha de aluminio de 0.40mm de espesor con un área de 50cmx50cm que costó $16.50. Para hacer la toma de datos en tiempo real, fue necesario una cámara de movimiento Kinect V1 que actualmente tiene un costo de $30 en el mercado y su respectivo adaptador para computadoras con un costo de $20. Para el control de los servomotores,

se adquirió un Arduino Mega por el precio de $34. Por último, para alimentar los servomotores del brazo robótico fue necesario conseguir dos adaptadores de voltaje de 5V para suministrar suficiente energía al sistema de movimiento. Cada uno de estos costó $4.60 dándonos como resultado un costo total de $149.70 para el apartado físico. En la **Tabla 3.2**, se puede apreciar de mejor forma el desglose de estos valores.

Tabla 3.2

Cotización de elementos físicos

Componente	**Cantidad**	**Precio unitario (USD)**	**Total (USD)**
Kit ensamble brazo	1	$40	$40
Base metálica	1	$16.50	$16.50
Kinect V1	1	$30	$30
Adaptador Kinect	1	$20	$20
Arduino MEGA	1	$34	$34
Adaptador de 5V	2	$4.60	$9.20
Total			$149.70

En cuanto al software utilizado y licencias, se emplearon tres distintos ambientes para el procesamiento los cuales fueron Arduino IDE, Proccesing y Matlab, de los cuales solo Matlab requiere una licencia pagada para poder utilizarlo ya que los demás son programas de software libre, es decir, no tienen costo alguno. La licencia de Matlab de forma perpetua para estudiantes actualmente tiene un costo de $55, sin embargo, si se quiere adquirir para uso académico su costo se eleva a los $550, por lo que todo radica en la intencionalidad con la que se vaya a realizar el proyecto.

Capítulo 4

4.1 Conclusiones y recomendaciones

4.1.1 Conclusiones

Luego de realizar el ensamble mecánico del robot, la captura de datos en tiempo real, el procesamiento de ángulos corporales, el desarrollo de controladores para cada articulación y un post procesamiento de datos para definir las posiciones angulares de los servomotores, hemos llegado a las siguientes conclusiones:

- El algoritmo de detección visual desarrollado logra capturar de forma inmediata cada uno de los movimientos del usuario ya que, gracias a la incorporación del dispositivo Kinect v1, se dispone de una capacidad de captura de 30 cuadros por segundo, permitiendo apreciar cada cambio en las articulaciones de una persona por más ligero que estos sean.
- El brazo robótico ensamblado posee gran estabilidad en reposo, eliminando cualquier tipo de vibraciones y movimientos involuntarios producidos por el peso de cada eslabón del sistema, presentando también una mejora en el transporte de toda la estructura física gracias a la geometría cuadrada y la robustez de la base acoplada.
- La utilización de la librería SimpleOpenNI para la captura de datos agilizó este proceso ya que incorporaba funciones específicas de detección de extremidades que trabajan en conjunto al dispositivo Kinect, acelerando el proceso de lectura y escritura de los ángulos creados por las posiciones límbicas de un usuario.
- El controlador de posición desarrollado para el proyecto contribuyó en gran medida al correcto funcionamiento del brazo robótico ya que redujo el sobre impulso existente en las articulaciones del robot hasta un máximo de 10%, evitó colisiones contra la propia estructura y minimizó los tiempos de estabilización, lo que es de vital importancia para un proceso de seguimiento en tiempo real, dando como resultado un máximo tiempo de estabilización de 0.4 segundos. Esto no solo mejoró el posicionamiento articular del robot, sino que también eliminó

cualquier movimiento errático que podría producirse, así como las vibraciones existentes en todas las articulaciones.

4.1.2 Recomendaciones

Tras culminar lo planificado en la propuesta y evaluar los resultados obtenidos, se presentan las siguientes recomendaciones primordiales para optimizar aún más el rendimiento del sistema:

- Aunque el controlador ha mejorado significativamente el comportamiento del sistema considerar afinar aún más los parámetros del controlador para reducir el sobre impulso, si es posible. Esto no solo aumentará la precisión del sistema, sino que también podría mejorar la eficiencia general del robot en tareas industriales. Una reducción adicional del sobre impulso puede traducirse en movimientos más precisos y controlados, lo que es esencial para aplicaciones que requieren alta exactitud y consistencia en el rendimiento.
- Realizar pruebas de robustez simulando diversas condiciones de carga y perturbaciones. Es crucial asegurarse de que el controlador mantenga su efectividad en diferentes escenarios operativos y que la estabilidad del sistema se preserve bajo condiciones variables. Estas pruebas permitirán verificar cómo responde el robot a cambios inesperados o a la introducción de fuerzas externas, asegurando así que el sistema sigue siendo fiable y seguro en un entorno industrial dinámico y exigente.
- Analizar el consumo energético del robot con el nuevo controlador en funcionamiento. Dado que minimizar movimientos innecesarios reduce el uso de energía, optimizar el controlador para hacer los movimientos más eficientes puede llevar a un funcionamiento más económico en términos de energía, lo cual es fundamental en entornos industriales. Un análisis detallado del consumo energético puede revelar oportunidades para

ahorrar costos y mejorar la sostenibilidad del sistema, factores que son cada vez más importantes en la industria moderna.

- Aunque el robot se comporta de manera ideal en las condiciones actuales, ten en cuenta que las demandas operativas o las condiciones ambientales pueden cambiar. Debes estar preparado para implementar estrategias de control similares si los requisitos de rendimiento futuros lo exigen. La flexibilidad para adaptar y ajustar los controladores en función de nuevas necesidades garantizará que el robot pueda seguir cumpliendo con las expectativas, incluso en circunstancias que puedan evolucionar con el tiempo.

Referencias

[1] B. Siciliano and O. Khatib, Eds., *Springer handbook of robotics*. Springer, 2016.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[3] J. J. Lee and A. Behal, "Vision-guided robotic manipulation: A review," *IEEE Trans. Ind. Informat.*, vol. 8, no. 1, pp. 48-66, 2012. doi: 10.1109/TII.2011.2172453.

[4] D. Kragic and H. I. Christensen, "Survey on visual servoing for manipulation," *Computational Vision and Active Perception Laboratory*, pp. 1-12, 2002.

[5] M. A. Goodrich and A. C. Schultz, "Human-robot interaction: A survey," *Found. Trends Hum.-Comput. Interact.*, vol. 1, no. 3, pp. 203-275, 2007. doi: 10.1561/1100000005.

[6] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," IEEE MultiMedia, vol. 19, no. 2, pp. 4-10, Mar. 2012.

[7] K. Dautenhahn, "Socially intelligent robots: Dimensions of human-robot interaction," *Philos. Trans. R. Soc. B*, vol. 362, no. 1480, pp. 679-704, 2007. doi: 10.1098/rstb.2006.2004.

[8] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, planning and control*. Springer, 2010.

[9] P. Corke, *Robotics, vision and control: Fundamental algorithms in MATLAB*. Springer, 2011.

[10] E. Pavía Parra, Diseño, construcción y evaluación de un robot cartesiano XYZ electroneumático, Tesis de grado, Universidad Politécnica de Valencia, 2016. [Online]. Available: https://riunet.upv.es/bitstream/handle/10251/70548/Pav%C3%ADa%20Parra%20-

%20Dise%C3%B1o%2C%20construcci%C3%B3n%20y%20evaluaci%C3%B3n%20de%20un%20robot%20cartesiano%20XYZ%20electroneum%C3%A1tico.pdf?sequence=2&isAllowed=y .

[11] Huang, L., Dou, Y., Fan, Y., He, Z., Zhang, S., Zhang, L., & Zhang, B. (2022). Development of a robot position and attitude measuring system based on a laser tracker and an articulated arm coordinate machine. Metrology Science and Technology, 66(1), 26-31. https://doi.org/10.12338/j.issn.2096-9015.2021.0005

[12] Anónimo. Autonomía de los robots. http://community.fortunecity.ws/campus/essay/680/ANATOM%CDA.html

[13] Anónimo. Autonomía de los robots. http://community.fortunecity.ws/campus/essay/680/ANATOM%CDA.html

[14] ResearchGate. Robot manipulador SCARA. https://www.researchgate.net/figure/Figura-1-Robot-manipulador-SCARA_fig1_33514994

[15] Londoño, J. P. Maya, and J. Giraldo, "Aproximación al diseño de robots paralelos, análisis de caso del robot delta," Memorias de la Revista EIA, vol. 23, pp. 77-91, 2016. [Enlace]. Available: https://revistas.eia.edu.co/index.php/mem/article/view/833/751

[16] A. Alonso, "Ingeniería de sistemas y automática. Tema 2 Morfología," Universidad de Oviedo, 2011. [Enlace]. Available: http://www.isa.uniovi.es/~alonsog/Robotica/02%20Morfologia.pdf

[17] "Robots industriales: Tecnología y aplicaciones," Sicma21, Oct. 25, 2021. [Enlace]. Available: https://www.sicma21.com/robots-industriales-tecnologia-y-aplicaciones/#Robot_cartesiano

[18] "Robot de picking automático para final de línea," VM Systems, Apr. 14, 2021. [Enlace]. Available: https://vmsystems.es/2021/04/14/robot-de-picking-automatico-para-final-de-linea/

[19] "Movimientos de cámara: Ejemplos y definición," Aprendercine. [Enlace]. Available: https://aprendercine.com/movimientos-de-camara-ejemplos-definicion/

[20] Microsoft Corporation, "Especificaciones Kinect (Versión libre)." [Enlace]. Available: https://d1wqtxts1xzle7.cloudfront.net/31853932/Especificaciones_Kinect-libre.pdf?1392361187=&response-content-disposition=inline%3B+filename%3DEspecificaciones_Kinect.pdf&Expires=1718388805&Signature=S6p~jd99PlN-J~hJq0hTrgL49sifklWs7v1QkJxY5t7SfHe8rx6aVjXOQ1GkN8Gh77aIBRdSNC2FtNfZBnjs4fpx08jbT2HWMBGMFO8hZiSq0GcPJkHlV7FdR7C-PjE4fo1TLTJtg2G6SkYydNP3evIxkq4dmIoZ~SPLEf1IaY6oQ0divxZyZwvWJag5qYxpygHEqBrZ4pkahD7zUUzt2nWgZ59MKQBgRDnZ3cSRBvZgCtPex~nTn3BUqBMc0qqEfCMRVazxSrDit~wgpKmZDcW4YqRRSZih~XGzLKveoR6bj5Esx3surBwjWOU9FXU9SbygZdvaI~8K6J9YcjtYuA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

[21] University of KwaZulu-Natal, "Exploring the adoption of the Wii remote controller for teaching and learning in the physical sciences: A case study," *Journal of Information

Systems for Teacher Education and Health*, vol. 2, no. 1, pp. 45-56. [Enlace]. Available: https://journals.ukzn.ac.za/index.php/JISfTeH/article/view/652/1120.

[22] B. Fry and C. Reas, *Processing: A Programming Handbook for Visual Designers and Artists*. Cambridge, MA, USA: MIT Press, 2007.

[23] D. Shiffman, *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction*. Amsterdam, Netherlands: Morgan Kaufmann, 2008.

[24] M. Moler, MATLAB: A High-Performance Language for Technical Computing. Natick, MA, USA: MathWorks, 2016.

[25] M. Banzi and M. Shiloh, Getting Started with Arduino. Sebastopol, CA, USA: Maker Media, 2014.

[26 P. Aguayo, "Micro," 2004. [Enlace]. Available: https://d1wqtxts1xzle7.cloudfront.net/39407044/micro-libre.pdf?1445752291=&response-content-disposition=inline%3B+filename%3DMicro.pdf&Expires=1718584983&Signature=RtDYkWGufTB~7VWed-MKfAAztGIPFwGRhkjDoFpkilm05tpVqnxMo7TdNVuf6vHByaSq2KKjFQ3gsuwCZKXCED8MBYCGdMsWMcdp5piJ7ArLegMuDX6X72SAMKMJ12X5dqayUt~Z~4ACeQByBBs9g3jrbLGoKt-6QDrF5Fwn5QE-UyaoIQuOhdxXhQwJNx3-42qasnASskxZb9mMfX4XRuLpe5maZ4nwb45CCB2RHVOfwvYLzorQkpTNDbdAaOiD6NRMjfnUdfV2yjgfFDezOyZbu~ZDjCv-XKl055mft3bF0Fd0ai4ufDU2Vp9~nuVlLtNo4qz38yKzvmSt2mcmOA__.

[27] A. Myasischev, "Use of rescue mode for UAV on the basis of STM32 microcontrollers," ResearchGate, 2020. [Enlace]. Available: https://www.researchgate.net/profile/Alexandr-Myasischev/publication/343614866_Use_of_rescue_mode_for_UAV_on_the_basis_of_STM32_microcontrollers/links/5f34405ea6fdcccc43c595e1/Use-of-rescue-mode-for-UAV-on-the-basis-of-STM32-microcontrollers.pdf.

[28] M. George, "Pick and Place Robotic Arm Implementation Using Arduino," ResearchGate, 2017. [Enlace]. Available: https://www.researchgate.net/profile/Maria-George9/publication/316611175_Pick_and_Place_Robotic_Arm_Implementation_Using_Arduino/links/6011e716299bf1b33e2d413d/Pick-and-Place-Robotic-Arm-Implementation-Using-Arduino.pdf.

[29] S. Manjula and R. Karamagi, "Automatic Pick and Place Robot Manipulation Using a Microcontroller," ResearchGate, 2018. [Enlace]. Available: https://www.researchgate.net/profile/Robert-Karamagi/publication/328157770_Automatic_Pick_and_Place_Robot_Manipulation_Using_a_Microcontroller/links/5f61ef7292851c07896a519c/Automatic-Pick-and-Place-Robot-Manipulation-Using-a-Microcontroller.pdf.

[30] J. Fraden, Handbook of Modern Sensors: Physics, Designs, and Applications. Springer, 2010. [Enlace]. Available: https://books.google.com.ec/books?hl=es&lr=&id=wMm3BgAAQBAJ&oi=fnd&pg=PP1&dq=sensores+de+posicion&ots=6P3kezc6Wx&sig=d42jjQO9dBnVca75_0lCrRkJZhg&redir_esc=y#v=onepage&q=sensores%20de%20posicion&f=false.

[31] A. Menache, Understanding Motion Capture for Computer Animation. San Diego, CA, USA: Morgan Kaufmann, 2000.

[32] M. Gleicher, "Comparing Constraint-Based Motion Editing Methods," *Graphical Models*, vol. 63, no. 2, pp. 107-134, Mar. 2001.

[33] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," IEEE MultiMedia, vol. 19, no. 2, pp. 4-10, Mar. 2012.

[34] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, M. Cook, R. Moore, and A. Blake, "Real-Time Human Pose Recognition in Parts from Single Depth Images," in Proc. IEEE Conf. on Computer Vision and Pattern Recognition, Colorado Springs, CO, USA, Jun. 2011, pp. 1297-1304.

[35] F. Multon, L. France, A. Aubel, R. Kulpa, and J.-P. Cani, "Capturing and Animating Skinned Deformable Models in Real-time," in Proc. Computer Animation, Geneva, Switzerland, 1999, pp. 1-8.

[36] J. L. Mendoza Villarreal and A. Hernández Regalado, "Diseño e implementación de un controlador para un brazo robótico de 6 grados de libertad," in *Memorias del Congreso Nacional de Control Automático (CNCA)*, 2010. [Enlace]. Available: https://www.amca.mx/memorias/amca2010/Art%C3%ADculos/Rob%C3%B3tica/amca2010_submission_2.pdf.

[37] Digi-Key, "MG996R servo motor datasheet." [Enlace]. Available: https://www.digikey.com/en/htmldatasheets/production/5014637/0/0/1/mg996r.

[38] AliExpress, "Kit de brazo robótico." [Enlace]. Available: https://es.aliexpress.com/item/1005005606163486.html.

[39] ElektronikHobi, "Arduino Mega." [Enlace]. Available: https://elektronikhobi.net/arduino-kartlarini-taniyalim-1/arduino_mega/.

[40] J. Blum, Exploring Arduino: Tools and Techniques for Engineering Wizardry. New York, NY, USA: Wiley, 2019.

[41] A. Gilat, MATLAB: An Introduction with Applications. New York, NY, USA: Wiley, 2014.

[42] J. M. Mayor Torres, "Function description of Rehabilitation: Kinect+Shimmer sensors based application," Dec. 16, 2013. [Enlace]. Available: https://d1wqtxts1xzle7.cloudfront.net/34166506/descriptionmeiyorreport2013-libre.pdf?1404993801=&response-content-disposition=inline%3B+filename%3DFunction_description_of_Rehabilitation_K.pdf&Expires=1725323198&Signature=KJtfVTU71iUymtLJ4uD-j4iTnKlLBwAShwp2Tn7mGNrU3NTUP5EDPW~BQZvy5uBR-12EjdgA1xJeX0vtEUlDlY39skk~9KOWi3rYZbcuVXtSlM6BFVRFbweE4b0y6G7xCThV2P7ulfvNkciGzIN95VEIRLw2j8GaPChIklRo1XLkH6j2btfqpuqcxg8UeIzkadz4bRVvP-wdSKvzBuw4RIO8D-PdNd9MRnO4IZpD2UrR0W58xXBjLhWuF761B4FMysi8RzAO4CAi~kgXmQgwSPrmGgLZhdWrGv9Aod6BxggMWBgeamoTs4AUWAZ1S3br-vjezeFznY17KtYWKvfY9g__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

[43] M. E. Huber, A. L. Seitz, M. Leeser, and D. Sternad, "Validity and reliability of Kinect skeleton for measuring shoulder joint angles: a feasibility study," Physiotherapy, vol. 101,

no. 4, pp. 389-393, 2015. [Enlace]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0031940615037530.

[44] M. Miknis, R. Davies, P. Plassmann, and A. Ware, "Efficient Point Cloud Pre-processing using The Point Cloud Library," CORE, 2018. [Enlace]. Available: https://core.ac.uk/download/pdf/227092485.pdf.

[45] J. L. Mendoza, M. Hernández, and A. Martínez, "Implementación en tiempo real de un sistema de control para un brazo robótico," 2023. [Enlace]. Available: https://d1wqtxts1xzle7.cloudfront.net/106959311/401-libre.pdf?1698380311=&response-content-disposition=inline%3B+filename%3DImplementacion_en_tiempo_real_de_un_sist.pdf&Expires=1725323743&Signature=LCObRvLVIndj4xmxmiA-Cxn2LrnzLTwpaPPAqZzNamwIUadlWCs9-5pDijUNpC322KafeV6o43WW7WQSP8suL8M9BYo1QHH-IvSUgVlfk2hOeC5ARoU3WSXcZxnzWoVRFCvZ4H~2q-JbxFQgerNH9FTQvjBVPYBTWsYJa18fi38AtnETXxDk5KmN9v8yYMY3zT6q4bS0LGRbNZB~f0SCeVrpxzVOIdwI~eVsvhjOSFGCIgbksed-K6k9NhdFOBHGSU4ogjBxz74G6KU6V8aa08A-DHznm50jKDjwjU0pJvyQ72-MDS5I9xm3N3SFIj0Xa8SXbhVWic7K11nvbX-uMw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

[47] M. Álvarez, "Modelo matemático de un motor de corriente continua y aplicación a sistemas de control," 2016. [Enlace]. Available: https://d1wqtxts1xzle7.cloudfront.net/44838314/25_LAJPE_611_Manuel_Alvarez_preprint_corr_f-libre.pdf?1460950209=&response-content-

disposition=inline%3B+filename%3DModelo_matematico_de_un_motor_de_corrien.pdf&Expires=1725323888&Signature=ARwnHlW2b7zYVlQwdZ2xWlLNPXvLI87M0Glla2M2ImEZejYB2FpB16aQBlc~zjjcIR-BV8-wXQC79GexYKXbvogeMIAbWrBhozaw-ZA9BcYUvMZqW96-OmctdCZqUh9qqupkqUMcR2ftUOvYgE4D8TRKo2As0VvfVNsNqaymF0v0OoMsv13HfvdIhJ8Aaik15WusDtPkDNBibUu7lwxOXWQflWcsC~F4J1hbGp0GlEERVHCL-suGm9t8F9UIkqI45ZN~1TY8OZPSSug-5ChldstkxFFGTu0BOZIOikRrHpv059gn8-nym-x9pZtKEqdqIDzswe9PpqOFsHNI9mIlGA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

[48] F. Sciascio, "Control mioeléctrico de un brazo robótico," ResearchGate, 2010. [Enlace]. Available: https://www.researchgate.net/profile/Fernando-Sciascio/publication/255710718_Control_Mioelectrico_de_un_Brazo_Robotico/links/5e7ff951a6fdcc139c103d87/Control-Mioelectrico-de-un-Brazo-Robotico.pdf.

[49] V. Mosquera, "Control de fuerza por impedancia y estudio de un sensor de fuerza para una prótesis de mano de nueve grados de libertad," ResearchGate, 2020. [Enlace]. Available: https://www.researchgate.net/profile/Victor-Mosquera-2/publication/337033103_Control_de_fuerza_por_impedancia_y_estudio_de_un_sensor_de_fuerza_para_una_protesis_de_mano_de_nueve_grados_de_libertad/links/6320abf5071ea12e362ec38e/Control-de-fuerza-por-impedancia-y-estudio-de-un-sensor-de-fuerza-para-una-protesis-de-mano-de-nueve-grados-de-libertad.pdf.

Apéndice A: códigos de programación

Apéndice A.1: código de Processing para captura de ángulos articulares de un usuario usando Kinect V1

```
import SimpleOpenNI.*;
import processing.serial.*;

Serial port;
//Generate a SimpleOpenNI object
SimpleOpenNI kinect;

//Vectors used to calculate the center of the mass
PVector com = new PVector();
PVector com2d = new PVector();

//Up
float LeftshoulderAngle = 0;
float LeftelbowAngle = 0;
float RightshoulderAngle = 0;
float RightelbowAngle = 0;

//Legs
float RightLegAngle = 0;
float LeftLegAngle = 0;

//Timer variables
float a = 0;

void setup() {
    size(640, 480);
    kinect = new SimpleOpenNI(this);
    kinect.enableDepth();
```

```
	//kinect.enableIR();
	kinect.enableUser();// because of the version this change
	//size(640, 480);
	fill(255, 0, 0);
	//size(kinect.depthWidth()+kinect.irWidth(), kinect.depthHeight());
	kinect.setMirror(false);
	String portName = "COM9"; // Especifica el puerto COM9
	port = new Serial(this, portName, 9600); // Establece la velocidad de transmisión a 9600 baudios
	println("Conectado a " + portName);
}

void draw() {
	kinect.update();
	//image(kinect.depthImage(), 0, 0);
	//image(kinect.irImage(),kinect.depthWidth(),0);
	image(kinect.userImage(),0,0);
	IntVector userList = new IntVector();
	kinect.getUsers(userList);
	if (userList.size() > 0) {
		int userId = userList.get(0);
		//If we detect one user we have to draw it
		if( kinect.isTrackingSkeleton(userId)) {
			//DrawSkeleton
			drawSkeleton(userId);
			//drawUpAngles
			ArmsAngle(userId);
			//Draw the user Mass
			MassUser(userId);
			//AngleLeg
			LegsAngle(userId);
		}
```

```
        }
}
//Draw the skeleton
void drawSkeleton(int userId) {
        stroke(0);
        strokeWeight(5);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD, SimpleOpenNI.SKEL_NECK);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_LEFT_SHOULDER);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_LEFT_ELBOW);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,
SimpleOpenNI.SKEL_LEFT_HAND);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_RIGHT_SHOULDER);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_RIGHT_ELBOW);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
SimpleOpenNI.SKEL_RIGHT_HAND);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_LEFT_HIP);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,
SimpleOpenNI.SKEL_LEFT_KNEE);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,
SimpleOpenNI.SKEL_LEFT_FOOT);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_RIGHT_HIP);
```

```
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
SimpleOpenNI.SKEL_RIGHT_KNEE);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,
SimpleOpenNI.SKEL_RIGHT_FOOT);
        kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
SimpleOpenNI.SKEL_LEFT_HIP);
        noStroke();
        fill(255,0,0);
        drawJoint(userId, SimpleOpenNI.SKEL_HEAD);
        drawJoint(userId, SimpleOpenNI.SKEL_NECK);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_ELBOW);
        drawJoint(userId, SimpleOpenNI.SKEL_NECK);
        drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER);
        drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW);
        drawJoint(userId, SimpleOpenNI.SKEL_TORSO);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HIP);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_KNEE);
        drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HIP);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_FOOT);
        drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_KNEE);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HIP);
        drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_FOOT);
        drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HAND);
        drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HAND);
}

void drawJoint(int userId, int jointID) {
        PVector joint = new PVector();
        float confidence = kinect.getJointPositionSkeleton(userId, jointID,
                                        joint);
        if(confidence < 0.5) {
```

```
            return;
        }
        PVector convertedJoint = new PVector();
        kinect.convertRealWorldToProjective(joint, convertedJoint);
        ellipse(convertedJoint.x, convertedJoint.y, 5, 5);
}
//Generate the angle
float angleOf(PVector one, PVector two, PVector axis) {
        PVector limb = PVector.sub(two, one);
        return degrees(PVector.angleBetween(limb, axis));
}

//Calibration not required

void onNewUser(SimpleOpenNI kinect, int userID) {
        println("Start skeleton tracking");
        kinect.startTrackingSkeleton(userID);
}

void onLostUser(SimpleOpenNI curContext, int userId) {
        println("onLostUser - userId: " + userId);
}

void MassUser(int userId) {
        if(kinect.getCoM(userId,com)) {
                kinect.convertRealWorldToProjective(com,com2d);
                stroke(100,255,240);
                strokeWeight(3);
                beginShape(LINES);
                vertex(com2d.x,com2d.y - 5);
                vertex(com2d.x,com2d.y + 5);
                vertex(com2d.x - 5,com2d.y);
```

```
            vertex(com2d.x + 5,com2d.y);
            endShape();
            fill(0,255,100);
            text(Integer.toString(userId),com2d.x,com2d.y);
        }
}

public void ArmsAngle(int userId){
        // get the positions of the three joints of our right arm
        PVector rightHand = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_HAND,rightHand);
        PVector rightElbow = new PVector();

kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_ELBOW,rightElbow);
        PVector rightShoulder = new PVector();

kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_SHOULDER,rightShould
er);
        // we need right hip to orient the shoulder angle
        PVector rightHip = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_HIP,rightHip);
        // get the positions of the three joints of our left arm
        PVector leftHand = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_HAND,leftHand);
        PVector leftElbow = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_ELBOW,leftElbow);
        PVector leftShoulder = new PVector();

kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_SHOULDER,leftShoulder)
;
        // we need left hip to orient the shoulder angle
        PVector leftHip = new PVector();
```

```
kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_HIP,leftHip);
// reduce our joint vectors to two dimensions for right side
PVector rightHand2D = new PVector(rightHand.x, rightHand.y);
PVector rightElbow2D = new PVector(rightElbow.x, rightElbow.y);
PVector rightShoulder2D = new PVector(rightShoulder.x,rightShoulder.y);
PVector rightHip2D = new PVector(rightHip.x, rightHip.y);
// calculate the axes against which we want to measure our angles
PVector torsoOrientation = PVector.sub(rightShoulder2D, rightHip2D);
PVector upperArmOrientation = PVector.sub(rightElbow2D, rightShoulder2D);
// reduce our joint vectors to two dimensions for left side
PVector leftHand2D = new PVector(leftHand.x, leftHand.y);
PVector leftElbow2D = new PVector(leftElbow.x, leftElbow.y);
PVector leftShoulder2D = new PVector(leftShoulder.x,leftShoulder.y);
PVector leftHip2D = new PVector(leftHip.x, leftHip.y);
// calculate the axes against which we want to measure our angles
PVector torsoLOrientation = PVector.sub(leftShoulder2D, leftHip2D);
PVector upperArmLOrientation = PVector.sub(leftElbow2D, leftShoulder2D);
// calculate the angles between our joints for rightside
RightshoulderAngle = angleOf(rightElbow2D, rightShoulder2D, torsoOrientation);
RightelbowAngle = angleOf(rightHand2D,rightElbow2D,upperArmOrientation);
// show the angles on the screen for debugging
fill(255,0,0);
scale(1);
text("Right shoulder: " + int(RightshoulderAngle) + "\n" + " Right elbow: " +
int(RightelbowAngle), 20, 20);
// calculate the angles between our joints for leftside
LeftshoulderAngle = angleOf(leftElbow2D, leftShoulder2D, torsoLOrientation);
LeftelbowAngle = angleOf(leftHand2D,leftElbow2D,upperArmLOrientation);
// show the angles on the screen for debugging
fill(255,0,0);
scale(1);
```

```
        text("Left shoulder: " + int(LeftshoulderAngle) + "\n" + " Left elbow: " +
int(LeftelbowAngle), 20, 55);
        byte out[] = new byte[4];
        out[0] = byte(RightshoulderAngle);
        out[1] = byte(LeftshoulderAngle);
        out[2] = byte(RightelbowAngle);
        out[3] = byte(LeftelbowAngle);
        port.write(out);
}

void LegsAngle(int userId) {
        // get the positions of the three joints of our right leg
        PVector rightFoot = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_FOOT,rightFoot);
        PVector rightKnee = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_KNEE,rightKnee);
        PVector rightHipL = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_HIP,rightHipL);
        // reduce our joint vectors to two dimensions for right side
        PVector rightFoot2D = new PVector(rightFoot.x, rightFoot.y);
        PVector rightKnee2D = new PVector(rightKnee.x, rightKnee.y);
        PVector rightHip2DLeg = new PVector(rightHipL.x,rightHipL.y);
        // calculate the axes against which we want to measure our angles
        PVector RightLegOrientation = PVector.sub(rightKnee2D, rightHip2DLeg);
        // calculate the angles between our joints for rightside
        RightLegAngle = angleOf(rightFoot2D,rightKnee2D,RightLegOrientation);
        fill(255,0,0);
        scale(1);
        text("Right Knee: " + int(RightLegAngle), 500, 20);
        // get the positions of the three joints of our left leg
        PVector leftFoot = new PVector();
        kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_FOOT,leftFoot);
```

```
    PVector leftKnee = new PVector();
    kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_KNEE,leftKnee);
    PVector leftHipL = new PVector();
    kinect.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_HIP,leftHipL);
    // reduce our joint vectors to two dimensions for left side
    PVector leftFoot2D = new PVector(leftFoot.x, leftFoot.y);
    PVector leftKnee2D = new PVector(leftKnee.x, leftKnee.y);
    PVector leftHip2DLeg = new PVector(leftHipL.x,leftHipL.y);
    // calculate the axes against which we want to measure our angles
    PVector LeftLegOrientation = PVector.sub(leftKnee2D, leftHip2DLeg);
    // calculate the angles between our joints for left side
    LeftLegAngle = angleOf(leftFoot2D,leftKnee2D,LeftLegOrientation);
    // show the angles on the screen for debugging
    fill(255,0,0);
    scale(1);
    text("Leftt Knee: " + int(LeftLegAngle), 500, 55);
}
```

Apéndice A.2: código de MATLAB para crear modelo del brazo robótico, definir parámetros Kp y Ki y visualizar las respuestas a una entrada escalón de los sistemas.

```
 clc
% Datos de entrada y salida
input_angle1 = array_original1; % Angulos de entrada
output_angle1 = array_con_errores1; % Angulos de salida
timen1 = time1; % Tiempos correspondientes
% Crear el objeto iddata
data = iddata(output_angle1', input_angle1', timen1(2) - timen1(1)); % Crear iddata con los datos
% Definir el orden del modelo
% Orden de estado para el modelo de espacio de estados
order = 2; % Puedes ajustar este valor según el problema
% Estimar el modelo de espacio de estados usando mínimos cuadrados
sys_ss1 = ssest(data, order);
```

```
% Convertir el modelo de espacio de estados a función de transferencia
sys_tf1 = tf(sys_ss1);
% Mostrar la función de transferencia estimada
sys_tf1
% Visualizar la respuesta del sistema
figure;
step(sys_tf1);
title('Respuesta al Escalón del Sistema Estimado para Articulación 1');
controller_PI1 = pidtune(sys_tf1, 'PI'); % Controlador PI sintonizado para la primera articulación
figure;
closed_loop_sys1 = feedback(controller_PI1 * sys_tf1, 1);
closed_loop_sys1
step(closed_loop_sys1);
title('Respuesta al Escalón del Controlador para Articulación 1');

%%
clc
% Datos de entrada y salida
input_angle2 = array_original2; % Angulos de entrada
output_angle2 = array_con_errores2; % Angulos de salida
timen2 = time2; % Tiempos correspondientes
% Crear el objeto iddata
data = iddata(output_angle2', input_angle2', timen2(2) - timen2(1)); % Crear iddata con los datos
% Definir el orden del modelo
% Orden de estado para el modelo de espacio de estados
order = 2; % Puedes ajustar este valor según el problema
% Estimar el modelo de espacio de estados usando mínimos cuadrados
sys_ss2 = ssest(data, order);
% Convertir el modelo de espacio de estados a función de transferencia
sys_tf2 = tf(sys_ss2);
% Mostrar la función de transferencia estimada
sys_tf2
```

```
% Visualizar la respuesta del sistema
figure;
step(sys_tf2);
title('Respuesta al Escalón del Sistema Estimado para Articulación 2');
controller_PI2 = pidtune(sys_tf2, 'PI'); % Controlador PI sintonizado para la primera articulación
figure;
closed_loop_sys2 = feedback(controller_PI2 * sys_tf2, 1);
closed_loop_sys2
step(closed_loop_sys2);
title('Respuesta al Escalón del Controlador para Articulación 2');


%%
% Datos de entrada y salida
input_angle3 = array_original3; % Angulos de entrada
output_angle3 = array_con_errores3; % Angulos de salida
timen3 = time3; % Tiempos correspondientes
% Crear el objeto iddata
data = iddata(output_angle3', input_angle3', timen3(2) - timen3(1)); % Crear iddata con los datos
% Definir el orden del modelo
% Orden de estado para el modelo de espacio de estados
order = 2; % Puedes ajustar este valor según el problema
% Estimar el modelo de espacio de estados usando mínimos cuadrados
sys_ss3 = ssest(data, order);
% Convertir el modelo de espacio de estados a función de transferencia
sys_tf3 = tf(sys_ss3);
% Mostrar la función de transferencia estimada
sys_tf3
% Visualizar la respuesta del sistema
figure;
step(sys_tf3);
title('Respuesta al Escalón del Sistema Estimado para Articulación 3');
```

```
controller_PI3 = pidtune(sys_tf3, 'PI'); % Controlador PI sintonizado para la primera articulación
figure;
closed_loop_sys3 = feedback(controller_PI3 * sys_tf3, 1);
closed_loop_sys3
step(closed_loop_sys3);
title('Respuesta al Escalón del Controlador para Articulación 3');

%%
% Datos de entrada y salida
input_angle4 = array_original4; % Angulos de entrada
output_angle4 = array_con_errores4; % Angulos de salida
timen4 = time4; % Tiempos correspondientes
% Crear el objeto iddata
data = iddata(output_angle4', input_angle4', timen4(2) - timen4(1)); % Crear iddata con los datos
% Definir el orden del modelo
% Orden de estado para el modelo de espacio de estados
order = 2; % Puedes ajustar este valor según el problema
% Estimar el modelo de espacio de estados usando mínimos cuadrados
sys_ss4 = ssest(data, order);
% Convertir el modelo de espacio de estados a función de transferencia
sys_tf4 = tf(sys_ss4);
% Mostrar la función de transferencia estimada
sys_tf4
% Visualizar la respuesta del sistema
figure;
step(sys_tf4);
title('Respuesta al Escalón del Sistema Estimado para Articulación 4');
controller_PI4 = pidtune(sys_tf4, 'PI'); % Controlador PI sintonizado para la primera articulación
figure;
closed_loop_sys4 = feedback(controller_PI4 * sys_tf4, 1);
closed_loop_sys4
step(closed_loop_sys4);
```

```
title('Respuesta al Escalón del Controlador para Articulación 4');

%%

% Datos de entrada y salida
input_angle5 = array_original5; % Angulos de entrada
output_angle5 = array_con_errores5; % Angulos de salida
timen5 = time5; % Tiempos correspondientes
% Crear el objeto iddata
data = iddata(output_angle5', input_angle5', timen5(2) - timen5(1)); % Crear iddata con los datos
% Definir el orden del modelo
% Orden de estado para el modelo de espacio de estados
order = 2; % Puedes ajustar este valor según el problema
% Estimar el modelo de espacio de estados usando mínimos cuadrados
sys_ss5 = ssest(data, order);
% Convertir el modelo de espacio de estados a función de transferencia
sys_tf5 = tf(sys_ss5);
% Mostrar la función de transferencia estimada
sys_tf5
% Visualizar la respuesta del sistema
figure;
step(sys_tf5);
title('Respuesta al Escalón del Sistema Estimado para Articulación 5');
controller_PI5 = pidtune(sys_tf5, 'PI'); % Controlador PI sintonizado para la primera articulación
figure;
closed_loop_sys5 = feedback(controller_PI5 * sys_tf5, 1);
closed_loop_sys5
step(closed_loop_sys5);
title('Respuesta al Escalón del Controlador para Articulación 5');

%%
```

```
% Datos de entrada y salida
input_angle6 = array_original6; % Angulos de entrada
output_angle6 = array_con_errores6; % Angulos de salida
timen6 = time6; % Tiempos correspondientes
% Crear el objeto iddata
data = iddata(output_angle6', input_angle6', timen6(2) - timen6(1)); % Crear iddata con los datos
% Definir el orden del modelo
% Orden de estado para el modelo de espacio de estados
order = 1; % Puedes ajustar este valor según el problema
% Estimar el modelo de espacio de estados usando mínimos cuadrados
sys_ss6 = ssest(data, order);
% Convertir el modelo de espacio de estados a función de transferencia
sys_tf6 = tf(sys_ss6);
% Mostrar la función de transferencia estimada
sys_tf6
% Visualizar la respuesta del sistema
figure;
step(sys_tf6);
title('Respuesta al Escalón del Sistema Estimado para Articulación 6');
controller_PI6 = pidtune(sys_tf6, 'PI'); % Controlador PI sintonizado para la primera articulación
figure;
closed_loop_sys6 = feedback(controller_PI6 * sys_tf6, 1);
closed_loop_sys6
step(closed_loop_sys6);
title('Respuesta al Escalón del Controlador para Articulación 6');
```

Apéndice A.3: código de Arduino IDE para leer valores desde puerto serial y escribirlos en los servomotores de articulación del brazo robótico

```
//Libreria Servo
#include <Servo.h>
```

```
// Declara los 6 Servos
Servo rightshoulder;
Servo leftshoulder;
Servo rightelbow;
Servo leftelbow;
Servo rightknee;
Servo leftknee;
// Declara el Arreglo servoAngles con 6
// que seran los angulos de tus servos
int nextServo =    0;
int servoAngles[]    =    {0,0,0,0,0,0};
void setup()
{
// Conectas los servos a sus pines respectivos
rightshoulder.attach(9);
rightelbow.attach(10);
leftshoulder.attach(11);
leftelbow.attach(12);
rightknee.attach(13);
leftknee.attach(14);
//Se inicia el serial
Serial.begin(9600);
}
void loop()
{
 if (Serial.available() >= 4) {  // Verifica si hay al menos 4 bytes disponibles en el buffer serial
    int rs = Serial.read();  // Lee el primer byte y lo asigna a var1
    int ls = Serial.read();  // Lee el segundo byte y lo asigna a var2
    int re = Serial.read();  // Lee el tercer byte y lo asigna a var3
    int le = Serial.read();  // Lee el cuarto byte y lo asigna a var4
    int rk = Serial.read();  // Lee el quinto byte y lo asigna a var5
    int lk = Serial.read();  // Lee el sexto byte y lo asigna a var6
```

```
  rightshoulder.write(rs);
  righttelbow.write(constrain(re,10,150));
  leftshoulder.write(constrain(ls,10,160));
  leftelbow.write(constrain(le,20,160));
  rightknee.write(rk);
  leftknee.write(lk);

}
}
```

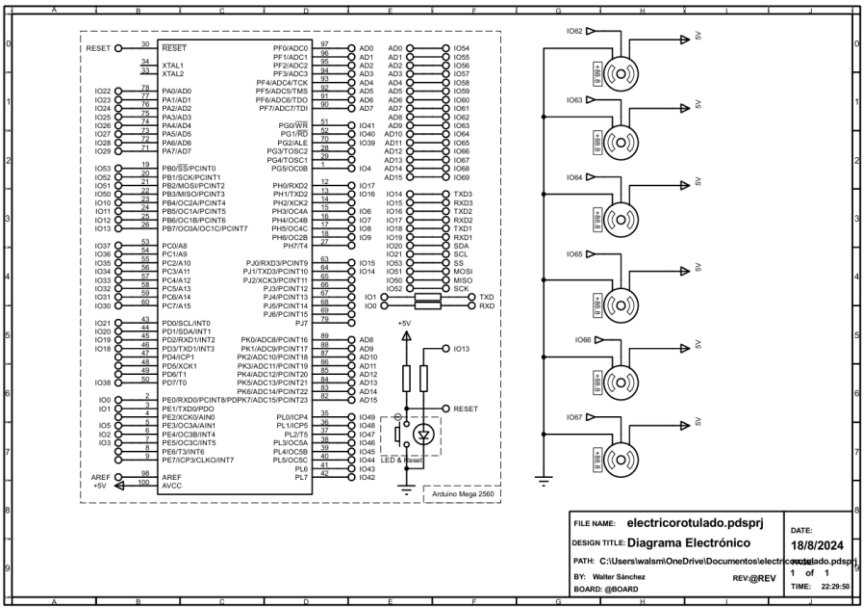
Apéndice A.4: código de Arduino IDE para obtener valores experimentales de las entradas y salidas de los servomotores para crear el modelo experimental

```
#include <Servo.h>

Servo miServo;  // Crear objeto Servo

const int totalAngulos = 181;  // 181 posiciones desde 0 hasta 180 grados
int angulosImpuestos[totalAngulos];
int angulosReales[totalAngulos];

void setup() {
  miServo.attach(9);  // Conectar el servomotor al pin 9
  Serial.begin(9600);

  for (int i = 0; i < totalAngulos; i++) {
    int anguloImpuesto = i;  // Ángulo impuesto
    miServo.write(anguloImpuesto);  // Mover el servomotor al ángulo impuesto
    delay(1000);  // Esperar 1 segundo para que el servomotor alcance la posición

    int anguloReal = miServo.read();  // Leer el ángulo real del servomotor
```

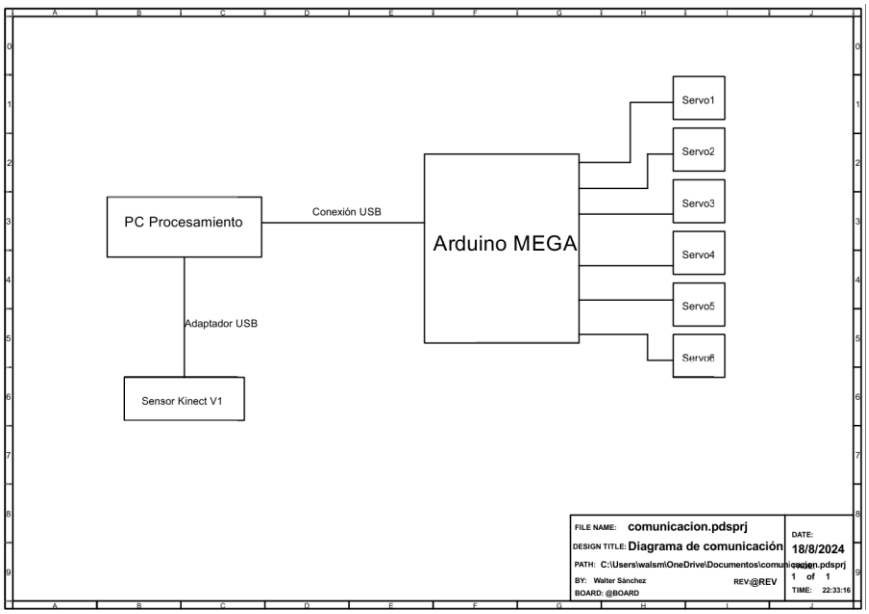
```
    angulosImpuestos[i] = anguloImpuesto;  // Guardar el ángulo impuesto
    angulosReales[i] = anguloReal;  // Guardar el ángulo real

    // Mostrar los valores en el monitor serie
    Serial.print("Ángulo Impuesto: ");
    Serial.print(angulosImpuestos[i]);
    Serial.print(" | Ángulo Real: ");
    Serial.println(angulosReales[i]);
  }
}

void loop() {
  // No se necesita código en loop() para este ejemplo
}
```

Apéndice B: planos mecánicos, eléctricos y de comunicación

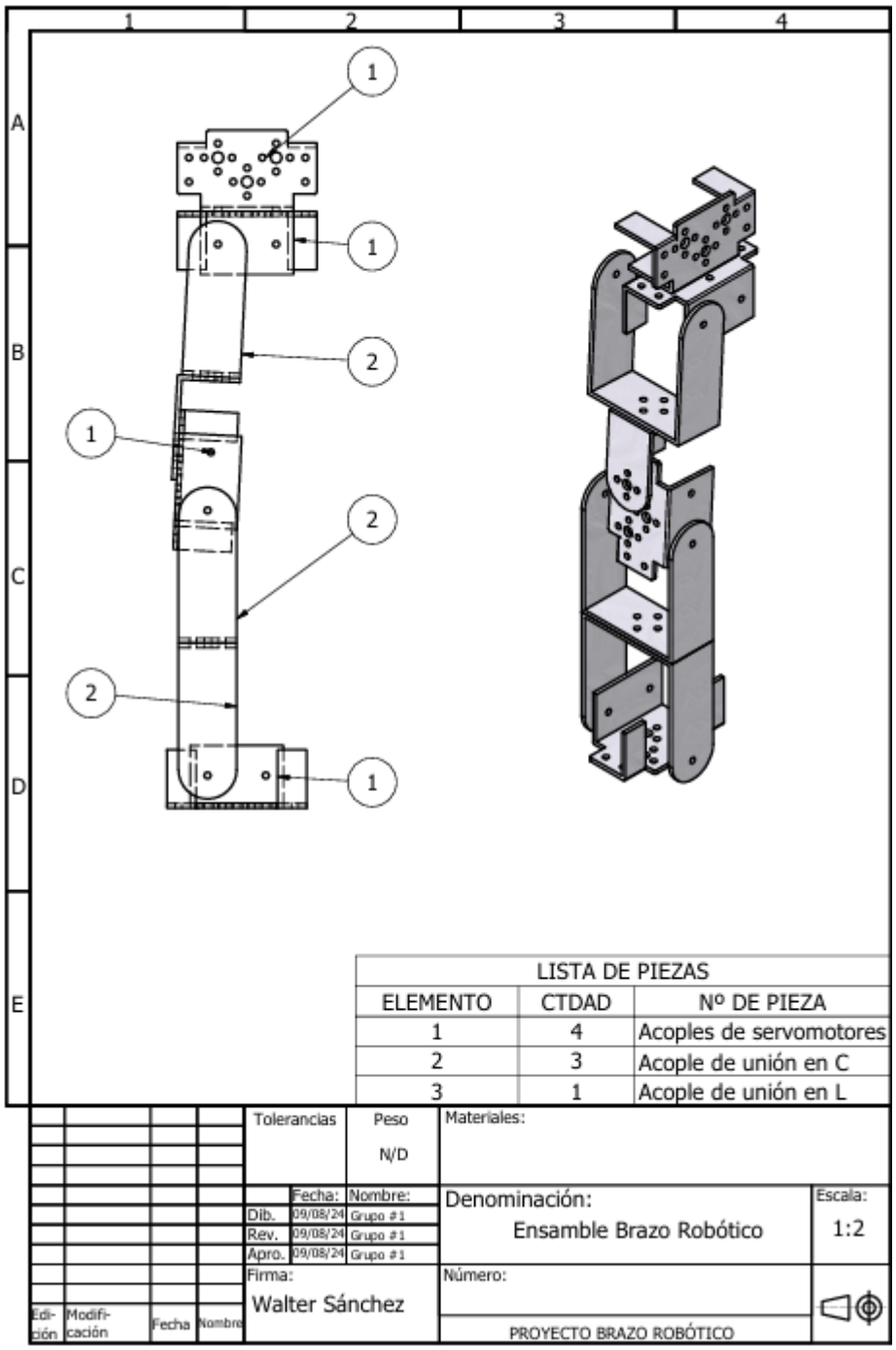
Apéndice B.1: plano eléctrico de conexión entre servomotores y placa Arduino Mega

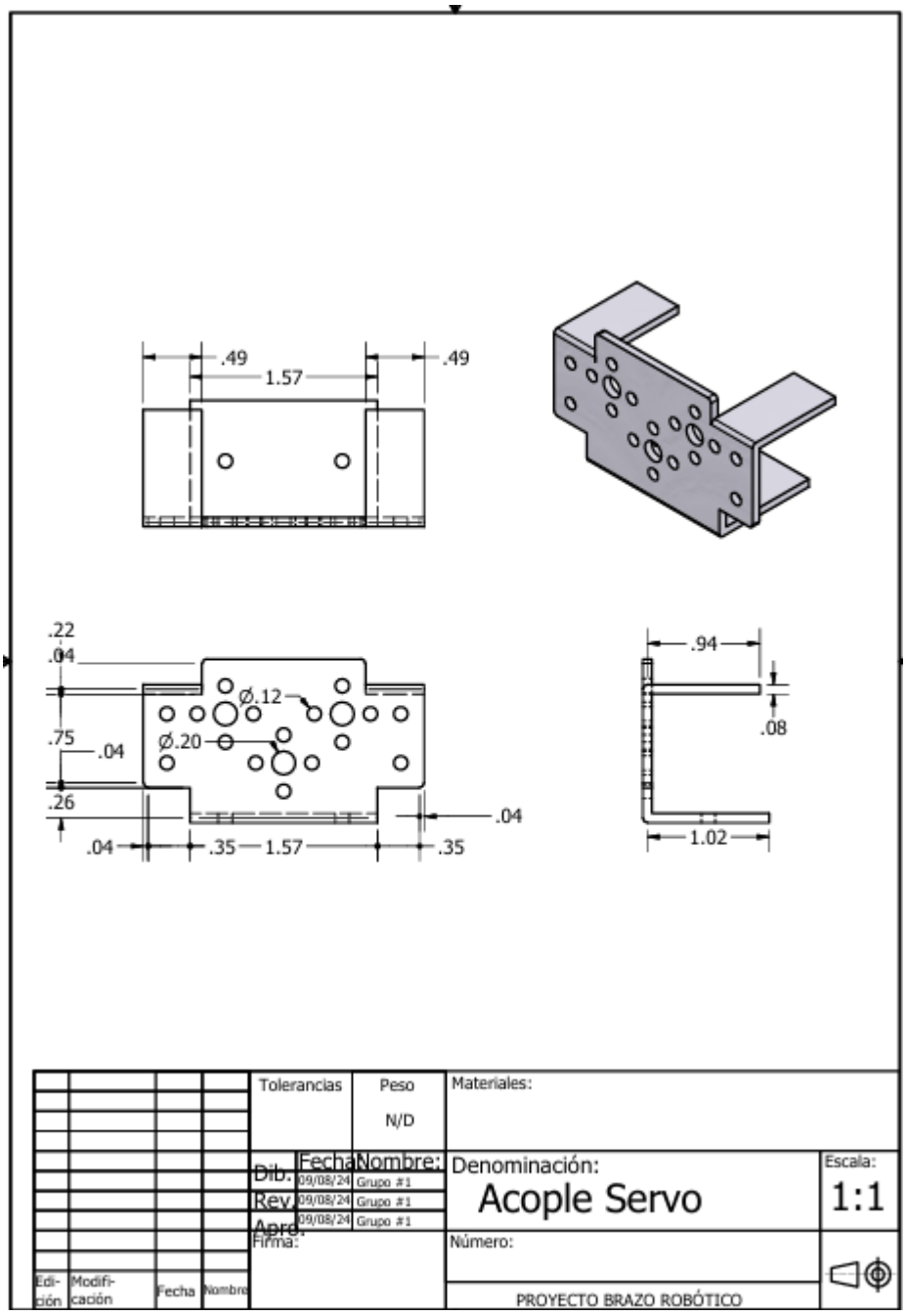


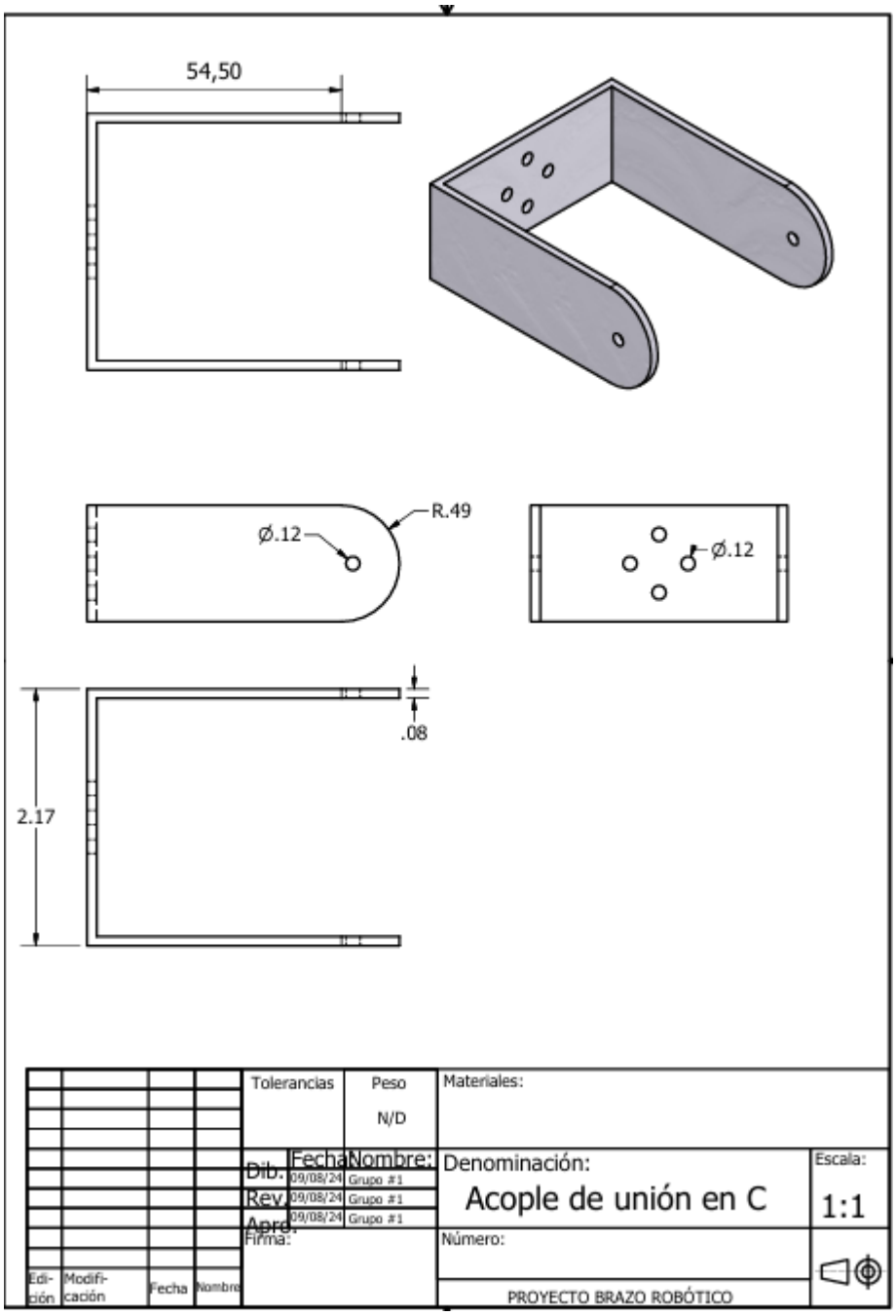
Apéndice B.2: plano de comunicación del sistema

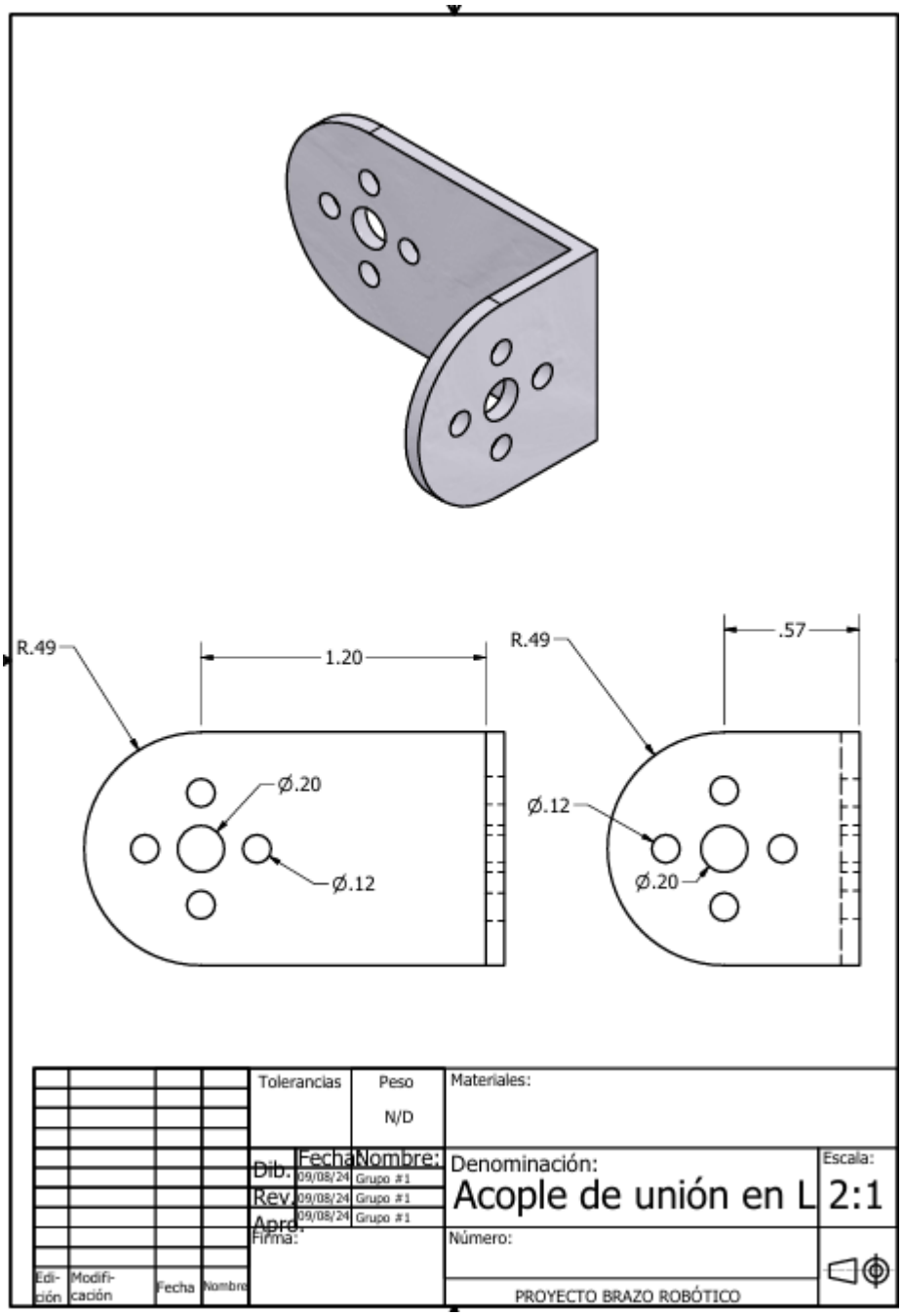


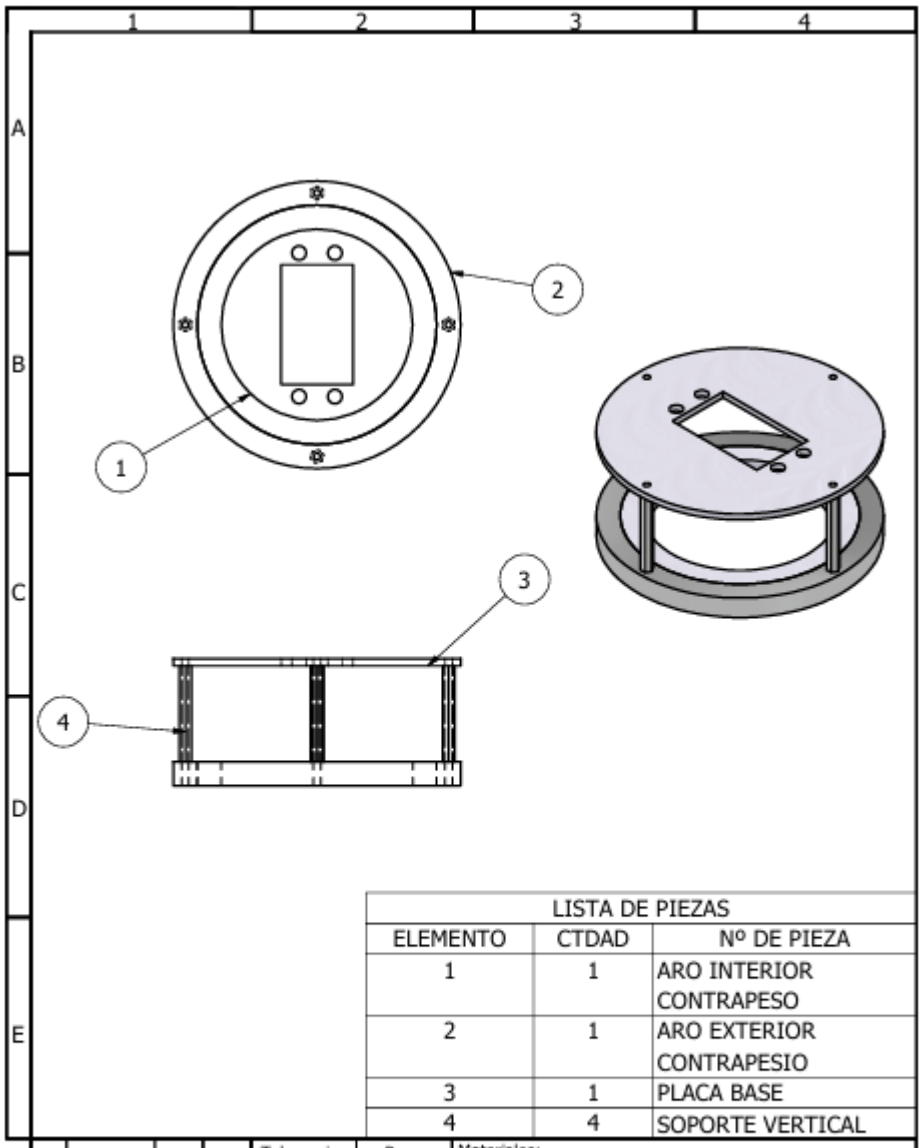
Apéndice B.3: planos mecánicos de las piezas y ensamble del brazo robótico




.49
1.57
.49
.22
.04
Ø.12
Ø.20
.75
.04
.26
.04
.04
.35
1.57
.35
.94
.08
1.02
Tolerancias
Peso
N/D
Materiales:
Dib.
Rev.
Aprb.
Fecha
Nombre:
09/08/24
Grupo #1
09/08/24
Grupo #1
09/08/24
Grupo #1
Firma:
Denominación:
Acople Servo
Escala:
1:1
Número:
PROYECTO BRAZO ROBÓTICO
Edi-ción
Modifi-cación
Fecha
Nombre


54,50
R.49
Ø.12
Ø.12
.08
2.17
Tolerancias
Peso
N/D
Materiales:
Dib.
Rev.
Aprd.
Fecha
Nombre:
09/08/24
Grupo #1
Firma:
Denominación:
Acople de unión en C
Escala:
1:1
Número:
PROYECTO BRAZO ROBÓTICO
Edi-ción
Modifi-cación
Fecha
Nombre

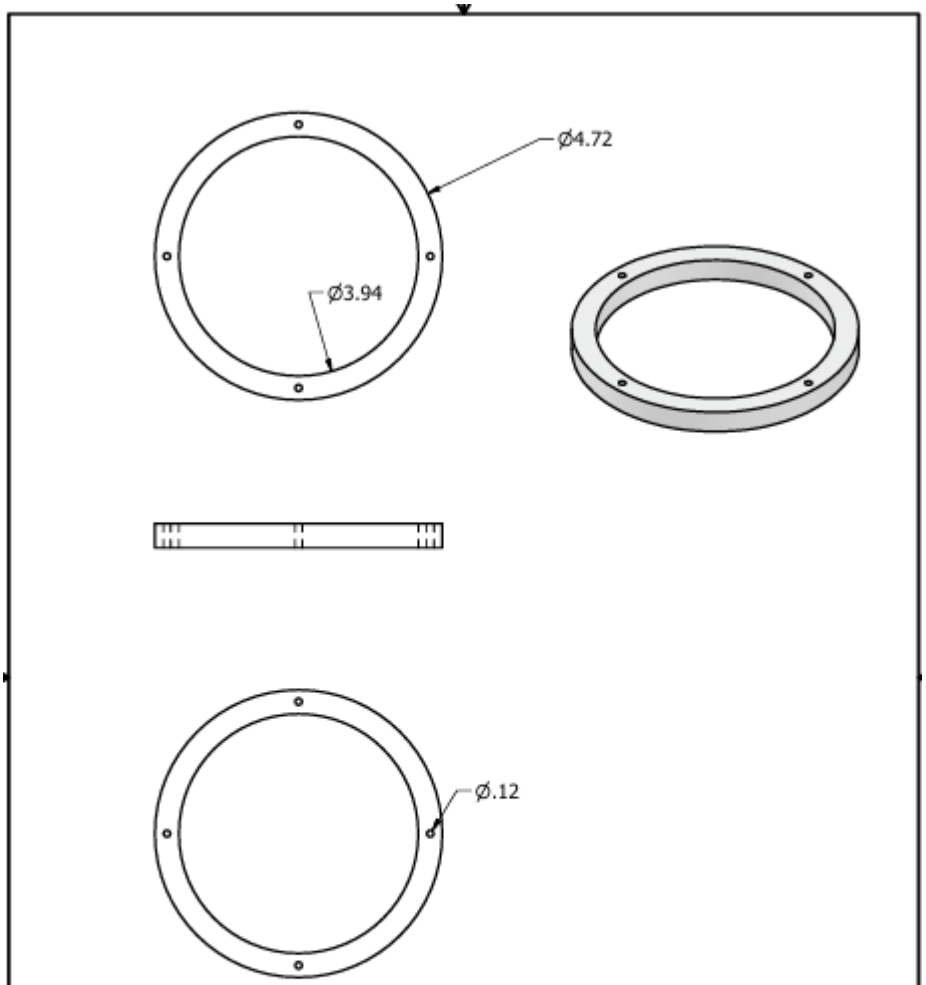

R.49
1.20
Ø.20
Ø.12
R.49
.57
Ø.12
Ø.20
Tolerancias
Peso
N/D
Materiales:
Dib.
Rev.
Apro.
Fecha
09/08/24
09/08/24
09/08/24
Nombre:
Grupo #1
Grupo #1
Grupo #1
Firma:
Denominación:
Acople de unión en L
Escala:
2:1
Número:
PROYECTO BRAZO ROBÓTICO
Edi-ción
Modifi-cación
Fecha
Nombre



LISTA DE PIEZAS		
ELEMENTO	CTDAD	Nº DE PIEZA
1	1	ARO INTERIOR CONTRAPESO
2	1	ARO EXTERIOR CONTRAPESIO
3	1	PLACA BASE
4	4	SOPORTE VERTICAL

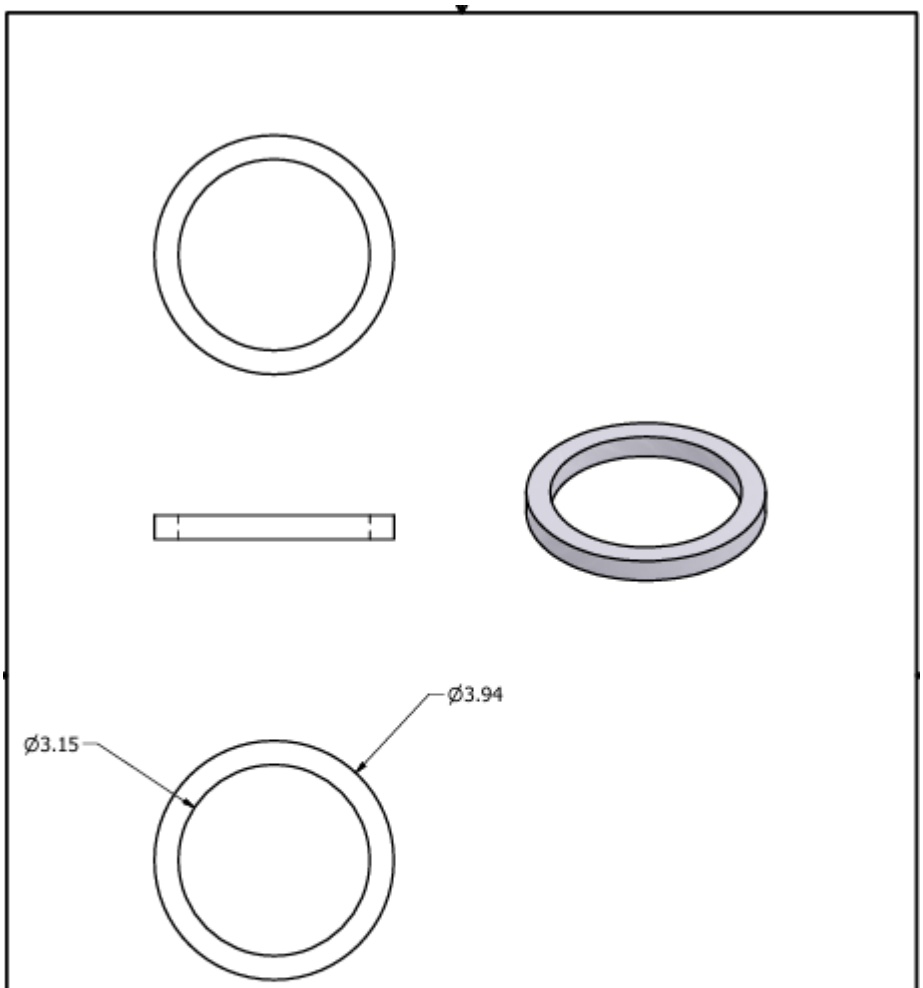
Tolerancias	Peso N/D	Materiales:	
	Fecha:	Nombre:	Denominación: Ensamble Base / Escala: 1:2
Dib.	09/08/24	Grupo #1	
Rev.	09/08/24	Grupo #1	
Apro.	09/08/24	Grupo #1	
Firma: Walter Sánchez			Número: / PROYECTO BRAZO ROBÓTICO

Edición	Modificación	Fecha	Nombre



				Tolerancias	Peso N/D	Materiales:
				Dib. 09/08/24 Grupo #1 Rev. 09/08/24 Grupo #1 Aprob. 09/08/24 Grupo #1	Fecha Nombre:	Denominación: Aro Exterior Base
				Firma:		Número:
Edición	Modificación	Fecha	Nombre			PROYECTO BRAZO ROBÓTICO

Escala: 1:2



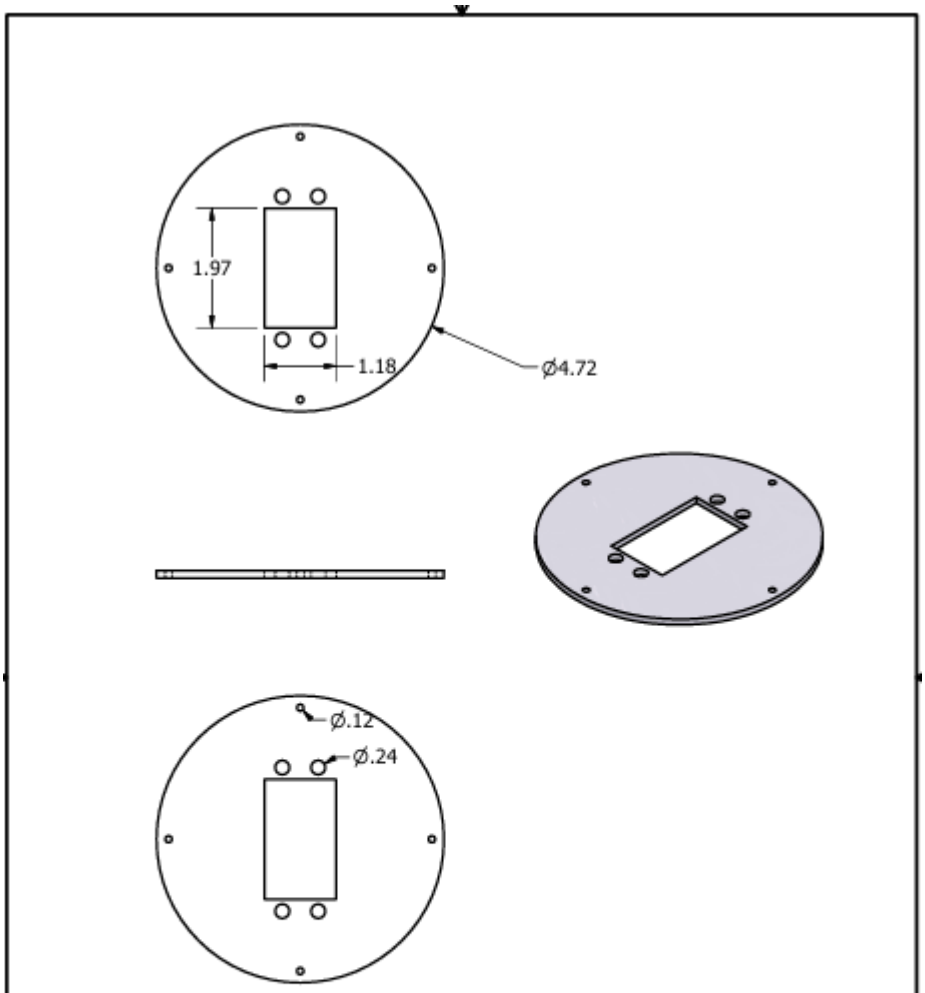
				Tolerancias	Peso N/D	Materiales:
				Dib.	Fecha 09/08/24	Nombre: Grupo #1
				Rev.	09/08/24	Grupo #1
				Apro.	09/08/24	Grupo #1
				Firma:		
Edición	Modificación	Fecha	Nombre			

Denominación: Aro Interior Base

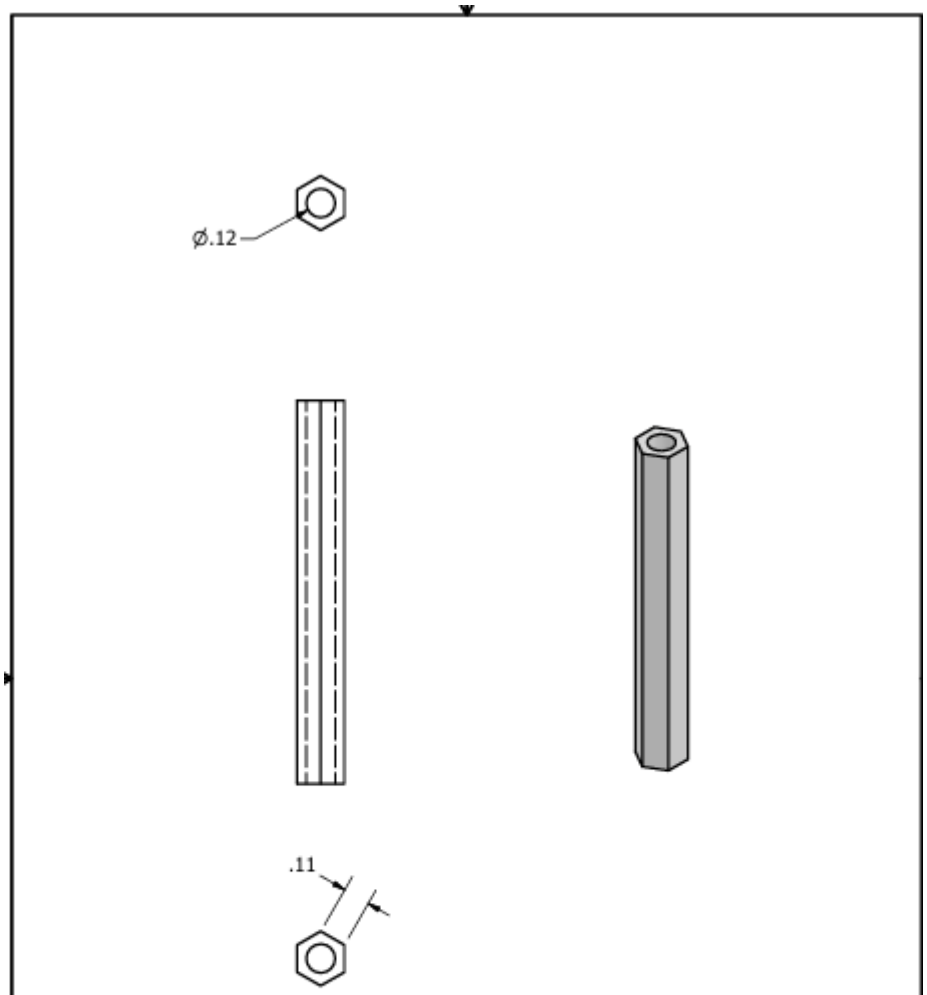
Escala: 1:2

Número:

PROYECTO BRAZO ROBÓTICO



				Tolerancias	Peso N/D	Materiales:
				Dib. Fecha 09/08/24 Nombre: Grupo #1	Denominación: Placa de la base	Escala: 1:2
				Rev. 09/08/24 Grupo #1		
				Aprb. 09/08/24 Grupo #1		
				Firma:	Número:	
Edición	Modificación	Fecha	Nombre		PROYECTO BRAZO ROBÓTICO	



				Tolerancias	Peso N/D	Materiales:
				Dib. Fecha 09/08/24 Nombre: Grupo #1	Denominación: Soporte vertical	Escala: 2:1
				Rev. 09/08/24 Grupo #1		
				Aprd. 09/08/24 Grupo #1		
				Firma:	Número:	
Edición	Modificación	Fecha	Nombre		PROYECTO BRAZO ROBÓTICO	